



## INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

<b>(51) International Patent Classification <sup>6</sup> :</b> <b>G06F 12/02</b>	<b>A1</b>	<b>(11) International Publication Number:</b> <b>WO 99/00730</b> <b>(43) International Publication Date:</b> 7 January 1999 (07.01.99)
<b>(21) International Application Number:</b> PCT/US98/13478 <b>(22) International Filing Date:</b> 25 June 1998 (25.06.98)  <b>(30) Priority Data:</b> 08/883,291 26 June 1997 (26.06.97) US  <b>(71) Applicant:</b> SUN MICROSYSTEMS, INC. [US/US]; 901 San Antonio Road, Palo Alto, CA 94043 (US).  <b>(72) Inventors:</b> TREMBLAY, Marc; Apartment 3, 801 Waverly Street, Palo Alto, CA 94301 (US). O'CONNOR, James, Michael; 1877 Slate Drive, Union City, CA 94587 (US). STEELE, Guy, L., Jr.; 9 Lantern Lane, Lexington, MA 02173 (US). VISHIN, Sanjay; Apartment 89, 1055 Manet Avenue, Sunnyvale, CA 94087 (US). AGESEN, Ole; 15 Rolling Ridge Road, Franklin, MA 02038 (US). HELLER, Steven; 8 Thomas Drive, Chelmsford, MA 01824 (US). WHITE, Derek, R.; 54 Dana Road, Reading, MA 01867 (US).  <b>(74) Agent:</b> KOESTNER, Ken, J.; Skjerven, Morrill, MacPherson, Franklin & Friel LLP, Suite 700, 25 Metro Drive, San Jose, CA 95110 (US).		<b>(81) Designated States:</b> JP, KR, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).  <b>Published</b> <i>With international search report.</i>
<b>(54) Title:</b> BOUNDED-PAUSE TIME GARBAGE COLLECTION SYSTEM AND METHOD INCLUDING WRITE BARRIER ASSOCIATED WITH A SOURCE INSTANCE OF A PARTIALLY RELOCATED OBJECT  <b>(57) Abstract</b>  A write barrier to stores into a partially relocated large or popular memory object facilitates bounded pause time implementations of relocating garbage collectors, including e.g., copying collectors, generational collectors, and collectors providing compaction. Such a write barrier allows a garbage collector implementation to interrupt relocation of large or popular memory objects so as to meet bounded pause time guarantees. A partially relocated object identifier store including "copy from" identifier storage accessible to write barrier logic allows the write barrier logic to maintain consistency between FromSpace and ToSpace instances of a partially relocated memory object. "Copy from" identifier storage allows the write barrier logic, or a trap handler responsive thereto, to broadcast a store-oriented memory access targeting the FromSpace instance to both FromSpace and ToSpace instances. Optional "How far" indication storage facilitates differentiation by the write barrier logic between a copied portion and an uncopied portion of the partially relocated memory object.		

*FOR THE PURPOSES OF INFORMATION ONLY*

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece	ML	Mali	TR	Turkey
BG	Bulgaria	HU	Hungary	MN	Mongolia	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MR	Mauritania	UA	Ukraine
BR	Brazil	IL	Israel	MW	Malawi	UG	Uganda
BY	Belarus	IS	Iceland	MX	Mexico	US	United States of America
CA	Canada	IT	Italy	NE	Niger	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NL	Netherlands	VN	Viet Nam
CG	Congo	KE	Kenya	NO	Norway	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NZ	New Zealand	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	PL	Poland		
CM	Cameroon	KR	Republic of Korea	PT	Portugal		
CN	China	KZ	Kazakhstan	RO	Romania		
CU	Cuba	LC	Saint Lucia	RU	Russian Federation		
CZ	Czech Republic	LI	Liechtenstein	SD	Sudan		
DE	Germany	LK	Sri Lanka	SE	Sweden		
DK	Denmark	LR	Liberia	SG	Singapore		
EE	Estonia						

**BOUNDED-PAUSE TIME GARBAGE COLLECTION SYSTEM AND METHOD INCLUDING  
WRITE BARRIER ASSOCIATED WITH A SOURCE INSTANCE OF A PARTIALLY RELOCATED  
OBJECT**

**BACKGROUND OF THE INVENTION**

**5    Field of the Invention**

The present invention relates to garbage collection, and in particular, to methods and apparatus for facilitating bounded pause time garbage collection.

**Description of the Related Art**

Traditionally, most programming languages have placed responsibility for dynamic allocation and  
10    deallocation of memory on the programmer. For example, in the C programming language, memory is  
allocated from the heap by the `malloc` procedure (or its variants). Given a pointer variable, `p`, execution of  
machine instructions corresponding to the statement `p=malloc(sizeof(SomeStruct))` causes  
pointer variable `p` to point to newly allocated storage for a memory object of size necessary for representing a  
`SomeStruct` data structure. After use, the memory object identified by pointer variable `p` can be  
15    deallocated, or freed, by calling `free(p)`. Pascal and C++ languages provide analogous facilities for  
explicit allocation and deallocation of memory.

Unfortunately, dynamically allocated storage may become unreachable if no reference, or pointer, to  
the storage remains in the set of root reference locations for a given computation. Memory objects that are no  
longer reachable, but have not been freed, are called *garbage*. Similarly, storage associated with a memory  
20    object can be deallocated while still referenced. In this case, a *dangling reference* has been created. In  
general, dynamic memory can be hard to manage correctly. In most programming languages, heap allocation  
is required for data structures that survive the procedure that created them. If these data structures are passed  
to further procedures or functions, it may be difficult or impossible for the programmer or compiler to  
determine the point at which it is safe to deallocate them.

25    Because of this difficulty, garbage collection, i.e., automatic reclamation of heap-allocated storage  
after its last use by a program, can be an attractive alternative model of dynamic memory management.  
Garbage collection is particularly attractive for functional languages, such as the JAVA™ (JAVA™ and  
JAVA-based trademarks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United  
States and other countries), Prolog, Lisp, Smalltalk, Scheme, Eiffel, Dylan, ML, Haskell, Miranda, Oberon,  
30    etc., which exhibit data sharing, delayed execution, and generally, less predictable execution orders than the  
procedural languages. See generally, Jones & Lins, *Garbage Collection: Algorithms for Automatic Dynamic  
Memory Management*, pp. 1-41, Wiley (1996) for a discussion of garbage collection and the classical  
algorithms therefor.

- 2 -

Three classical garbage collection methods are reference counting, mark-sweep, and copying storage reclamation. The first, reference counting, is based on maintaining a count of the number of references, e.g., pointers, to each memory object from active memory objects or root reference locations. When a new memory object is allocated and a pointer thereto is assigned, the memory object's reference count is set to one. Then, each time a pointer is set to refer to the memory object, the memory object's reference count is incremented. When a reference to the memory object is deleted or overwritten, the reference count is decremented. Memory objects with a reference count of zero are unreachable and can be collected as garbage. A reference counting garbage collector implementation typically includes an additional field, the *reference count*, in each memory object and includes incrementing and decrementing support as part of new object, delete object and update pointer functions.

In contrast, tracing collector methods involve traversal of reference chains through memory to identify live, i.e., referenceable, memory objects. One such tracing collector method is the mark-sweep method in which reference chains through memory are traversed to identify and mark live memory objects. Unmarked memory objects are garbage and are collected and returned to the free pool during a separate sweep phase. A mark-sweep garbage collector implementation typically includes an additional field, e.g., a *mark bit*, in each memory object. Mark-compact collectors add compaction to the traditional mark-sweep approach. Compaction relocates live objects to achieve beneficial reductions in fragmentation. Reference count methods may also employ compaction.

Another tracing method, copying collection, divides memory (or a portion thereof) into two semi-spaces, one containing current data and the other containing old data. Copying garbage collection begins by reversing the roles of the two semi-spaces. The copying collector then traverses the live objects in the old semi-space, *FromSpace*, copying reachable objects into the new semi-space, *ToSpace*. After all the live objects in *FromSpace* have been traversed and copied, a replica of the data structures exists in *ToSpace*. In essence, a copying collector scavenges live objects from amongst the garbage. A beneficial side effect of copying collection is that live objects are compacted into *ToSpace*, thereby reducing fragmentation.

Generational approaches build on the observations that (1) memory objects typically die young and that (2) tracing methods spend considerable resources traversing, copying, or relocating comparatively long-lived objects. Generational garbage collection schemes divide the heap into two or more generations, segregating objects by age, and concentrate collection efforts (or at least more vigorous collection efforts) on the younger generation(s). Since the youngest generation is small, garbage collection related pause times can, on average, be kept short. Garbage collection within a generation can be by either copying or mark-sweep collection. Promotion of memory objects from a younger to an older generation typically involves copying.

Because of the cost of copying large objects, some generational approaches have included large object areas. See e.g., Ungar and Jackson, *Tenuring Policies for Generation-based Storage Reclamation*, ACM SIGPLAN Notices, 23(11), pp. 1-17 (1988), Ungar and Jackson, *An Adaptive Tenuring Policy for*

- 3 -

*Generation Scavengers*, ACM Transactions on Programming Languages and Systems, 14(1), pp. 1-17 (1992). Typically, the technique is to separate a large object into a header portion stored in the generational part of the heap and a body portion stored in the large object area. The header portions are scavenged like other objects, but resources are not expended to copy the body portions. Ungar and Jackson found that pause times could be  
5 reduced by a factor of four by allocating 330 Kbytes to a large object area.

For interactive or real-time applications, the shortness of garbage collection pauses is an important figure of merit. Traditional implementations of tracing garbage collectors periodically interrupt execution of application programs in order to traverse memory in search of memory regions that are no longer in use. Unfortunately, hard real-time systems require worst-case guarantees that results be computed on-time. Even  
10 in mere interactive systems, pause time should be bounded and short. So-called incremental garbage collection methods attempt to avoid lengthy pauses caused by start and stop reclamation and instead interleave collection with application program cycles. To achieve this goal, an incremental collector must coherently manage heap accesses by the collector and application program (often more generally referred to as a mutator). Concurrent collectors, e.g., in a multiprocessor, present similar, though more stringent fine grain  
15 synchronization requirements.

In general, interleaved or concurrent relocating collectors present multiple-readers, multiple writers synchronization problems. Both read and write barrier methods have been used to prevent a mutator from disrupting garbage collection by altering the connectivity of the memory object referencing graph in a way that interferes with the collector's traversal thereof. See e.g., Steele, *Multiprocessing Compactifying Garbage  
20 Collection*, Communications of the ACM, 18(9), pp. 495-508 (1975) (write barrier, mark-sweep-compact collection); Dijkstra et al., *On-the-fly Garbage Collection: An Exercise in Cooperation*, Communications of the ACM, 21(11), pp. 965-975 (1978) (write barrier); Kung & Song, *An Efficient Parallel Garbage Collection System and its Correctness Proof*, IEEE Symposium on Foundations of Computer Science, pp. 120-131 (1977) (write barrier); Baker, *List Processing in Real-time on a Serial Computer*, Communications of the  
25 ACM, 21(4), pp. 280-93 (1978) (read barrier, copying collector); Brooks, *Trading Data Space for Reduced Time and Code Space in Real-time Garbage Collection on Stock Hardware*, in Conference Record of the 1984 ACM Symposium on Lisp and Functional Programming, Austin, Texas, pp. 256-62 (1984) (write barrier, copying collector); and Dawson, *Improved Effectiveness from a Real-time Lisp Garbage Collector*, Conference Record of the 1992 ACM Symposium on Lisp and Functional Programming, San Francisco,  
30 California, pp. 159-67 (1982) (write barrier, copying collector).

The Symbolics 3600 provided hardware read-barrier and write-barrier support for Baker style copying collection and for trapping intergenerational pointer stores. See Moon, *Architecture of the Symbolics 3600*, Proceedings of the 12th Annual International Symposium on Computer Architecture, pp. 76-83 (1985). The MIT Lisp machine and TI Explorer also provided hardware read-barrier support for Baker style copying  
35 collection. Nilsen and Schmidt describe a garbage collected memory module which implements hardware read- and write-barriers in U.S. Patent No. 5,560,003.

- 4 -

In addition to the fundamental challenge of managing mutator accesses to prevent alterations to the connectivity of the memory object referencing graph in a way that would interfere with the collector's traversal thereof, bounded pause time relocating collectors should address the substantial period of time required to relocate a large and/or popular memory object. A garbage collector can ensure that memory objects can be completely relocated within a bounded interval, by relegating large objects incompatible with the bounded interval to a large object area which may be collected by a non-relocating method (*see e.g.*, Ungar and Jackson, *Tenuring Policies for Generation-based Storage Reclamation*, ACM SIGPLAN Notices, 23(11), pp. 1-17 (1988)) or may be uncollected. However, lazy or incremental copying approaches are preferable.

Baker's solution was to include an additional link word in the header of large objects. In a FromSpace instance of the object, the link word stored a forwarding address to a ToSpace instance of the object. In the ToSpace instance of the object, the link word stored a backward link to the FromSpace instance. After the forwarding and backward links are set, the object was copied incrementally. Like small objects, fields of the large object were copied and scanned for pointers back into FromSpace objects that had not yet been copied. Baker used the scanned/unscanned boundary, defined by a garbage collection variable, *scan*, to steer write accesses to the partially copied large object. The cost of Baker's scheme, apart from an additional header word was a software write-barrier to writes into the ToSpace instance of the large object. If the address was greater than *scan*, the write was performed in the OldSpace instance using the backward link; otherwise the write was performed in the NewSpace instance.

Nilsen and Schmidt presented a hardware solution based on Baker's copying collector. In particular, Nilsen and Schmidt provided a garbage collected memory module in which a hardware read-barrier maintains the illusion that collection is complete. A hardware arbiter in the garbage collected memory module provided a hardware Baker-type read-barrier, and in addition provided read and write barriers to accesses into an as yet uncopied portion of an object instance in ToSpace. The garbage collected memory module maintained memory address registers to indicate the beginning and end of the uncopied portion. Read and write accesses to the uncopied portion were directed to the FromSpace instance.

#### SUMMARY OF THE INVENTION

Accordingly, it has been discovered that a write barrier to stores into a partially relocated large and/or popular memory object facilitates bounded pause time implementations of relocating garbage collectors, including e.g., copying collectors, generational collectors, and collectors providing compaction. Partially relocated object identifier storage and a write barrier responsive thereto allows relocation of large and/or popular memory objects by a garbage collector implementation to be interrupted so as to meet bounded pause time guarantees to a mutator process.

A partially relocated object identifier store including "copy from" identifier storage accessible to write barrier logic allows the write barrier logic to maintain consistency between FromSpace and ToSpace

- 5 -

instances of a partially relocated memory object. In some embodiments, the write barrier traps to a partially relocated object trap handler which maintains consistency. In other embodiments which include "copy from" identifier storage accessible to write barrier logic, the write barrier itself maintains consistency without software trap handler overheads. Optional "how far" indication storage facilitates differentiation by the write barrier logic, or by the partially relocated object trap handler, between a copied portion and an uncopied portion of the partially relocated memory object. Some embodiments may not differentiate between copied and uncopied portions of the partially relocated object. Furthermore, although "Copy to" and "Copy From" identifier storage accessible to the write barrier advantageously allows the write barrier logic to maintain consistency between FromSpace and ToSpace instances of a partially relocated memory object without software trap handler overhead, some embodiments may include such a software trap handler. A variety of alternative allocations of functionality between write barrier logic and a partially relocated object trap handler are envisioned and fall within the scope of the claims.

#### **BRIEF DESCRIPTION OF THE DRAWINGS**

The present invention may be better understood, and its numerous objects, features, and advantages made apparent to those skilled in the art by referencing the accompanying drawings.

Figures 1a and 1b (collectively referred to herein as Figure 1) are a block diagram of an exemplary embodiment of a virtual machine hardware processor that includes support for bounded pause-time garbage collector implementations in accordance with this invention.

Figure 2 depicts "builds upon" relationships between software and hardware components of a JAVA application environment including hardware processor (Figure 1) and software components of an exemplary JAVA virtual machine implementation.

Figure 3 illustrates several possible add-ons to the hardware processor of Figure 1.

Figures 4a and 4b (collectively referred to herein as Figure 4) depict one embodiment of partially relocated object identifier store and an exemplary memory object referencing graph in accordance with an semispace memory organization for a copying garbage collector prior to copying.

Figure 5 depicts the partially relocated object identifier store and semispace memory organization of Figure 4 during copying of a large object from FromSpace to ToSpace.

Figure 6 depicts accesses of a copying collector process and of a mutator process to FromSpace and ToSpace portions of a collected memory area that includes a partially relocated large object. Figure 6 illustrates operation of a write barrier of the hardware processor of Figure 1 in conjunction with the partially relocated object identifier store of Figures 4 and 5.

- 6 -

Figure 7 depicts an embodiment of a bounded-pause time garbage collection system including a barrier to stores to a copy-from instance of a partially relocated object and a trap handler therefor.

Figure 8 depicts an embodiment of a bounded-pause time garbage collection system including a hardware barrier to stores to a copy-from instance of a partially relocated object and including a partially  
5 relocated object identifier store having copy-from and copy-to instance identifiers. The embodiment of Figure 8 selectively broadcasts stores to both the copy-from instance and copy-to instance of the partially relocated object.

Figure 9 depicts an embodiment of a bounded-pause time garbage collection system including a hardware barrier to stores to a copy-from instance of a partially relocated object and including a partially  
10 relocated object identifier store having a copy-from instance identifier, a copy-to instance identifier, and a copied/uncopied boundary identifier. The embodiment of Figure 9 selectively directs or broadcasts stores depending on the partial relocation state of the partially relocated object to allow incremental copying.

Figure 10 depicts an embodiment of a bounded-pause time garbage collection system including barriers to stores to copy-from and copy-to instances of a partially relocated object and including a partially  
15 relocated object identifier store having copy-from and copy-to instance identifiers. The embodiment of Figure 10 allows both incremental copying and incremental pointer update.

Figure 11 depicts an embodiment of a bounded-pause time garbage collection system including barriers to stores to copy-from and copy-to instances of a partially relocated object and including a partially  
20 relocated object identifier store having a copy-from instance identifier, a copy-to instance identifier, a copied/uncopied boundary identifier, and a garbage collection phase indicator. The embodiment of Figure 11 selectively directs or broadcasts stores depending on the partial relocation state of the partially relocated object and garbage collection phase to allow both incremental copying and incremental pointer update.

Figure 12 depicts an embodiment of a bounded-pause time garbage collection system including barriers to loads from a copy-from instance of a partially relocated object and to stores to the copy-from  
25 instance and including a partially relocated object identifier store having a copy-from instance identifier, a garbage collection phase indicator, and a partially relocated object trap handler. The embodiment of Figure 12 selectively redirects or broadcasts stores and selectively redirects loads to allow both incremental copying and incremental pointer update, and in some variations, overlapping copy-from and copy-to instances.

Figure 13 depicts an embodiment of a bounded-pause time garbage collection system including  
30 barriers to loads from a copy-from instance of a partially relocated object and to stores to the copy-from instance and including a partially relocated object identifier store having a copy-from instance identifier, a copy-to instance identifier, and a garbage collection phase indicator. The embodiment of Figure 13 selectively redirects or broadcasts stores and selectively redirects loads to allow both incremental copying and incremental pointer update.



- 7 -

Figure 14 depicts an embodiment of a bounded-pause time garbage collection system including barriers to loads from a copy-from instance of a partially relocated object and to stores to the copy-from instance and including a partially relocated object identifier store having a copy-from instance identifier, a copy-to instance identifier, a copied/uncopied boundary identifier, and a garbage collection phase indicator.

5 The embodiment of Figure 13 selectively redirects stores and selectively redirects loads to allow both incremental copying, incremental pointer update, and overlapping copy-from and copy-to instances.

Figure 15 depicts operation of a bounded pause time garbage collection system allowing overlapping FromSpace and ToSpace portions of a partially relocated large object.

Figure 16 depicts an object reference (*objectref*) format in accordance with an embodiment of this  
10 invention.

Figure 17A depicts an object format in accordance with an embodiment of this invention.

Figure 17B depicts an alternative handled object format in accordance with an embodiment of this invention.

The use of the same reference symbols in different drawings indicates similar or identical items.

## 15 DESCRIPTION OF THE PREFERRED EMBODIMENT(S)

The following sets forth a detailed description of the best contemplated mode for carrying out the invention. The description is intended to be illustrative of the invention and should not be taken to be limiting.

Architectural support for bounded pause time implementations of relocating garbage collectors includes storage to identify locations from which (and in some embodiments, to which) a large and/or popular  
20 memory object is being relocated. As used herein, a large object is any memory object of size, structure, or content such that relocation of the complete memory object, potentially including updates of references thereto, may, under worst case conditions, be incompatible with bounded pause time guarantees of a garbage collector implementation. Embodiments in accordance with the present invention will exhibit varying degrees of conservatism with respect to operative definitions of a large object. For example, some embodiments may  
25 employ a size threshold, whereas others may include reference counts in the large object calculus. Still other embodiments, particularly those based substantially on hardware, may exploit the architectural support described herein without regard to memory object size. As used herein, a popular object is any memory object having a reference count such that relocation of the memory object, including updates of references thereto, may, under worst case conditions, be incompatible with bounded pause time guarantees of a garbage collector  
30 implementation. As above, embodiments in accordance with the present invention will exhibit varying degrees of conservatism with respect to operative definitions of a popular object.

- 8 -

In general, embodiments in accordance with the present invention may employ architectural support described herein for relocating large objects, popular objects, large and popular objects, all objects (including but not limited to large and/or popular objects), etc. Although such architectural support may be provided in hardware, in software, or in a combination of hardware and software, embodiments in which the architectural support is provided substantially in hardware will typically provide both increased performance and reduced memory requirement advantages. For this reason, an exemplary hardware processor embodiment is described herein. However, based on this description, those of skill in the art will appreciate alternative embodiments which fall within the scope of the claims which follow.

As used herein, relocating garbage collector is any garbage collector which as part of its operation relocates memory objects, including e.g., copying collectors, generational collectors, and collectors providing compaction or object relocation to reduce fragmentation and/or improve locality. Real-time or bounded pause time implementations of such relocating garbage collectors will typically be implemented as an incremental garbage collection software process whose computations are interleaved with user process activity. However, those of skill in the art will recognize concurrent garbage collection implementations based on the description herein. Furthermore, those of skill in the art will recognize suitable modifications, including provision of storage to identify multiple partially relocated objects, to support parallel collection implementations.

#### A JAVA Virtual Machine Instruction Processor Embodiment

Figure 1 depicts an exemplary hardware embodiment of a virtual machine instruction processor 100, hereinafter hardware processor 100, that includes support for bounded pause time relocating garbage collection in accordance with the present invention, and that directly executes processor architecture independent JAVA virtual machine instructions. The performance of hardware processor 100 in executing virtual machine instructions is typically better than high-end CPUs, such as the Intel PENTIUM microprocessor or the Sun Microsystems ULTRASPARC processor, (All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc., in the United States and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc. PENTIUM is a trademark of Intel Corp. of Sunnyvale, CA) interpreting the same virtual machine instructions with a software JAVA interpreter or with a JAVA just-in-time (JIT) compiler. In addition, hardware processor 100 is low cost and exhibits low power consumption. As a result, hardware processor 100 is well suited for portable applications.

Because hardware processor 100 provides a JAVA virtual machine instruction processor implementation substantially in hardware, 25-50 Kilobytes (Kbytes) of memory storage, e.g., read-only memory or random access memory, otherwise required by a software interpreter can be eliminated or alternatively allocated. Hardware support for garbage collection provides further advantages for a limited memory JAVA virtual machine implementation by reducing in-line code for garbage collection (e.g., compiler supplied read and/or write barrier support), by facilitating improved utilization of limited memory, and by

- 9 -

reducing garbage collection overheads and pause times. In environments where the expense of a large memory is prohibitive, including, for example, an Internet chip for network appliances, a cellular telephone processor, other telecommunications integrated circuits, or other low-power, low-cost applications such as embedded processors, and portable devices, hardware processor 100 is advantageous.

5 Even in environments where large memory is viable, hardware support for garbage collection reduces overheads associated with barrier implementations, facilitates improved utilization of memory, and reduces pause times for relocating garbage collector implementations. In particular, hardware processor 100 provides advantages for garbage collection methods and implementations in the context of an exemplary JAVA virtual machine implementation. However, based on the description herein, those of skill in the art will recognize  
10 variations for other JAVA virtual machine implementations, including e.g., interpreted and JIT compiler JAVA virtual machine implementations, as well as for other non-JAVA virtual machine implementations.

As used herein, a virtual machine is an abstract computing machine that, like a real computing machine, has an instruction set and uses various memory areas. A virtual machine specification defines a set of processor architecture independent virtual machine instructions that are executed by a virtual machine  
15 implementation. In general, a virtual machine implementation may be in hardware (e.g., as in the case of hardware processor 100), in software (e.g., as in the case of interpreted and JIT compiler implementations), or in hardware and software. Each virtual machine instruction defines a specific operation that is to be performed. The virtual machine need not understand the computer language that is used to generate virtual machine instructions or the underlying implementation of the virtual machine. Only a particular format for  
20 virtual machine instructions needs to be understood. In an exemplary embodiment, the virtual machine instructions are JAVA virtual machine instructions. Each JAVA virtual machine instruction includes one or more bytes that encode instruction identifying information, operands, and any other required information.

In this embodiment, hardware processor 100 (Fig. 1) processes the JAVA virtual machine instructions, which include bytecodes. Hardware processor 100 directly executes most of the bytecodes.  
25 However, execution of some of the bytecodes is implemented via microcode. Lindholm & Yellin, *The JAVA™ Virtual Machine Specification* (Addison-Wesley, 1996), ISBN 0-201-63452-X, which is incorporated herein by reference in its entirety, includes an exemplary set of JAVA virtual machine instructions. The particular set of virtual machine instructions supported by a hardware processor 100 is not an essential aspect of this invention. However, in view of the virtual machine instructions, those of skill in the art can modify the  
30 invention for a particular set of virtual machine instructions, or for changes to the JAVA virtual machine specification.

In one embodiment, hardware processor 100 includes an I/O bus and memory interface unit 110, an instruction cache unit 120 including instruction cache 125, an instruction decode unit 130 including non-quick to quick translator cache 131, a unified execution unit 140, a stack management unit 150 including stack cache  
35 155, a data cache unit 160 including data cache 165, and program counter and trap control logic 170. Support

- 10 -

for garbage collection features described herein resides primarily in integer unit 142 and register 144 of execution unit 140 with some additional support in program counter and trap control logic 170 (including e.g., support for forcing the program counter to a next JAVA virtual machine instruction following a trapping store). Each of these units is described below. In addition, an exemplary embodiment of hardware processor 5 100 is described in greater detail in Published PCT International Application No. PCT/US97/01221, entitled, "INSTRUCTION FOLDING FOR A STACK-BASED MACHINE," naming James Michael O'Connor and Marc Tremblay as inventors, the entirety of which is hereby incorporated by reference.

Figure 2 depicts a "builds upon" relationship between software and hardware components of a JAVA application environment such as, for example, an application environment partially defined by and partially 10 executable on hardware processor 100 (Fig. 1). JAVA application/applet software 210 exploits software components defining an applet/application programming interface 220 including AWT classes 241, net and I/O classes 242, and JAVA OS windows 243, JAVA OS graphics 248, TCP 244, NFS 245, UDP 246, IP 247, Ethernet 222, keyboard 249, and mouse 221 software components, which in one embodiment include JAVA bytecodes. In the embodiment of Figure 2, JAVA OS graphics 248 and Ethernet 222 software components 15 also include extended bytecodes beyond those defined by the baseline JAVA Virtual Machine Specification. Components of an embedded application programming interface (EAPI) 230 include foundation classes 231 and hardware and software components of JAVA virtual machine implementation 250 in accordance with the JAVA Virtual Machine Specification.

JAVA virtual machine implementation 250 includes hardware processor 100 and trap code 20 executable thereon to evaluate JAVA virtual machine instructions. In addition, JAVA virtual machine implementation 250 includes hardware support for extended bytecodes (including e.g., pointer store bytecodes and memory access barriers described below in the context of garbage collection); class loader 252, byte code verifier 253, thread manager 254, and garbage collector 251 software, and microkernel 255. JAVA virtual machine implementation 250 includes a JAVA virtual machine specification compliant portion 250a as well as 25 implementation dependent portions. Although the JAVA virtual machine specification specifies that garbage collection be provided, the particular garbage collection method employed is implementation-dependent.

Architectural features for garbage collection described herein in the context of an exemplary hardware processor 100 embodiment of JAVA virtual machine implementation 250 are particularly adapted for generational garbage collection methods. However, based on this description, those of skill in the art will 30 recognize the application of bounded-pause time support of this invention to relocating collectors in general, including e.g., non-generational collector implementations, incremental mark-compact collectors, copying collectors, etc.

Figure 3 illustrates several possible add-ons to hardware processor 100 to create more complicated system. Circuits supporting any of the eight functions shown, i.e., NTSC encoder 501, MPEG 502, Ethernet

- 11 -

controller 503, VIS 504, ISDN 505, I/O controller 506, ATM assembly/reassembly 507, and radio link 508 can be integrated into the same chip as hardware processor 100 of this invention.

In addition, those of skill in the art will appreciate a wide variety of computer systems incorporating hardware processor 100, including embodiments of hardware processor 100 with any of the above-described add-on circuits. An exemplary computer system embodiment includes physical memory storage (e.g., RAM and/or ROM), computer readable media access devices (e.g., disk, CD-ROM, tape, and/or memory technology based computer readable media access devices, etc.), input/output device interfaces (e.g., interfaces for keyboard and/or pointing devices, for display devices, etc.), and communications devices and/or interfaces. Suitable communications devices and/or interfaces include those for network- or telephony-based communications, for interfacing with communications networks including land-line and/or wireless portions of a public switched network, private networks, etc. In some embodiments of this invention, instruction streams (including e.g., JAVA bytecodes) are transmitted and/or received for execution by hardware processor 100 via such communications devices or interfaces.

#### Architectural Support for Garbage Collection

Hardware processor 100 provides hardware support for a variety of garbage collection methods, including relocating collector methods implemented as garbage collection software executable thereon. In particular, hardware processor 100 includes a partially relocated object identifier store and barrier support. In some embodiments such barrier support includes write barrier support for programmer selectable filtering of stores and/or support for execution-time resolution of store bytecodes to pointer-specific bytecodes to facilitate pointer store trapping, each as described in greater detail in PCT International Application No. PCT/US98/07622, entitled "GENERATION ISOLATION SYSTEM AND METHOD FOR GARBAGE COLLECTION," the entirety of which is hereby incorporated by reference.

#### Partially Relocated Object Identifier Store

Figure 4A depicts one embodiment of a partially relocated object identifier store 410 including from field 411, to field 412 and howfar field 413. Figure 4B depicts FromSpace 420 and ToSpace 430 portions of memory storage in accordance with a copying collector method. In a generational collector implementation, FromSpace 420 and ToSpace 430 portions may be semi-spaces of a single generation or may be portions of young and old generation spaces respectively. Alternatively, FromSpace 420 and ToSpace 430 portions may be semi-spaces in a non-generational collection space. Furthermore, FromSpace 420 and ToSpace 430 portions may overlap, may individually be contiguous or non-contiguous, are not necessarily of fixed size or have fixed locations in the memory storage. Additionally, in some embodiments, multiple FromSpace and ToSpace portions are supported with suitable modifications to partially relocated object identifier store 410 and to the barriers described herein. Figure 4B depicts a first referencing state of live memory objects A, B, C and large object 450 prior to copying into ToSpace 430. Root pointer set 440 is representative of any root

- 12 -

pointer set into the referencing structure, including e.g., pointers which originate from entries of operand stack and/or local variables storage represented in stack cache 155.

Figure 5 depicts a second referencing state of partially relocated object identifier store 410, FromSpace 420, and ToSpace 430 at interruption of large object 450 relocation. Live memory objects A and B have been copied to corresponding instances A' and B' in ToSpace 430. Upon interruption of the copying, large object 450 includes a copied portion 551 and a not completely copied portion 552. Contents of the copied portion 551 of large object 450 have been copied to a corresponding portion 551a of a ToSpace 430 instance of large object 450. ToSpace 430 storage for the remaining not completely copied portion of large object 450 is shown as portion 552a. From field 411 identifies large object 450 in FromSpace 420, to field 412 identifies the corresponding partially relocated instance 450a of large object 450, and howfar field 413 identifies a boundary between copied portion 551 and not completely copied portion 552 of large object 450. An overlapping copy direction field (not shown) is optional. In one embodiment, such a copy direction field includes a copy direction bit whose state indicates a forward or backward direction for copying of overlapped FromSpace and ToSpace instances. In other embodiments, copy direction is derived from the relative values of from field 411 and to field 412, as will be appreciated by those of skill in the art.

Howfar field 413 indicates the progress of large object copying and/or enables write barrier hardware and a partially relocated object trap handler to appropriately handle mutator accesses to a partially copied large object. In the embodiment of Figure 5, howfar field 413 indicates the address in memory of the boundary between copied portion 551 and uncopied portion 552. Nonetheless, those of skill in the art will recognize, based on this description, a variety of suitable alternative encodings including, e.g., an index off of the memory location indicated by from field 411, an indication of a corresponding boundary in the ToSpace instance 550a of large object 450, etc. Howfar field 413 encodes any such suitable indication.

Referring now to Figure 6, partially relocated object identifier store 410 is shown in the context of mutator process 610 and garbage collector process 620. In the embodiment of Figure 6, mutator process 610 and garbage collector process 620 are each defined by JAVA virtual machine instructions executable by hardware processor 100. Garbage collector process 620 includes any relocating collector. Although the description which follows is for the semi-space copying collector described above with reference to Figure 5, those of skill in the art will recognize the applicability to other relocating collectors. In the course of copying large object 450 from FromSpace 420 to ToSpace 430, garbage collector process 620 reads from and writes to collected space 630.

In one embodiment, garbage collector process 620 updates from field 411 and to field 412 at the beginning of large object 450 copying and updates howfar field 413 during copying such that, upon interruption by mutator process 610, howfar field 413 indicates the boundary between copied portion 551 and uncopied portion 552. In embodiments for concurrent mutator process 610 and garbage collector process 620 execution (e.g., in a multiprocessor), locking of howfar field 413 is provided and suitable methods therefor

- 13 -

will be recognized by those of skill in the art. Alternative embodiments may optionally forgo incremental updating of howfar field 413 in favor of an update on interruption.

5 Mutator process 610 reads from and writes to memory which includes collected space 630. In one embodiment, hardware processor 100 includes integer unit 142 support for a barrier 660 which compares object referencing information (e.g., an *objectref*) to contents of from field 411, to field 412, or both. A match with contents of from field 411 indicates that the referenced object is the FromSpace 420 instance of partially relocated large object 450. A match with contents of to field 412 indicates that the referenced object is the partially copied ToSpace 430 instance 450a. In some embodiments, evaluation of the particular field offset into the referenced object is compared against contents howfar field 413 to refine barrier 660 handling of  
10 copied and uncopied portions of the referenced object.

In various embodiments more particularly described below, barrier 660 includes write barrier support responsive to partially relocated object identifier store 410. In some embodiments, barrier 660 also includes read barrier support responsive to partially relocated object identifier store 410. In some embodiments, barrier 660 includes hardware read and/or write barrier support for trapping to a software partially relocated object  
15 trap handler which appropriately handles the trapping access (e.g., by broadcasting a write access, redirecting a write access, redirecting a read access, or allowing an access to complete normally). In other embodiments, a hardware read and/or write barrier itself provides such appropriate handling without software trap handler overhead.

In some embodiments, bounded pause time relocation is provided by elements of barrier 660 that  
20 allow incremental copying of a large and/or popular object while pointer updates thereto are via an object referencing handle. In such embodiments, pointer updates to even popular objects are trivially provided by update of a single pointer, i.e., of the object referencing handle. In other embodiments, bounded pause time relocation provided by elements of barrier 660 includes support for incremental updating of pointers to a popular, and possibly large, object. As used herein, bounded pause time relocation includes both copying and  
25 pointer update. Those of skill in the art will appreciate the applicability of embodiments described herein to bounded pause time relocation of large objects, to bounded pause time relocation of popular objects, and to bounded pause time relocation of large, popular objects. Handled and unhandled object referencing is described below in greater detail.

Depending on the garbage collection method(s) supported by a particular embodiment of hardware  
30 processor 100, write and/or read barrier support may be provided to prevent mutator process 610 from interfering with garbage collector process 620 by altering connectivity of the memory object referencing graph in a way that interferes with the collectors traversal thereof. For example, in one embodiment of hardware processor 100, programmer selectable hardware write barriers to intergenerational pointer stores, to all pointer stores, and to all stores (including support for filtered barrier variants of each) are provided as described in  
35 greater detail in the above-incorporated PCT International Application No. PCT/US98/07622, entitled,

- 14 -

"GENERATION ISOLATION SYSTEM AND METHOD FOR GARBAGE COLLECTION." In one embodiment of barrier 660, support for such additional barriers is integrated with the above-described barrier to stores into a partially relocated large object.

In embodiments in accordance with Figure 1, a partially relocated object trap handler includes JAVA  
 5 virtual machine instruction bytecodes executable on hardware processor 100 and initiated by program counter and trap control logic 170. Garbage collection traps, including e.g., a partially relocated object trap, are triggered before the trapping write is itself evaluated. Therefore, in order to prevent the processor from trapping infinitely, partially relocated object trap handler 650 emulates the trapping write along with the additional garbage collection-related functions. In one embodiment, program counter and trap control logic  
 10 170 then forces the program counter to the next JAVA virtual machine instruction following the trapping write. Based on the more detailed description which follows, those of skill in the art will recognize a variety of valid behaviors for a partially relocated object trap handler depending on consistency policies enforced.

Although garbage collection has been described generally above with respect to the illustrative Figure 6 embodiment of partially relocated object identifier store 410 including from field 411, to field 412,  
 15 howfar field 413, and in the context of a barrier 660 potentially including a partially relocated object trap handler, a variety of alternative embodiments are also suitable. In many of these alternative embodiments, a partially relocated object trap handler can be eliminated and mutator process accesses can be properly handled by hardware barrier support, thereby reducing overhead associated with a trap handler. Based on the description of exemplary embodiments which follows, those of skill in the art will appreciate a variety of  
 20 additional combinations and variations which fall within the scope of the claims.

Write Barrier Associated with Copy-From Instance Identifier Store

Figure 7 depicts an embodiment having a Copy-From register field with an associated write barrier. In particular, the embodiment of Figure 7 includes from field 411 of partially relocated object identifier store 410 (see Fig. 4), hardware write barrier 740, and partially relocated object trap handler 750. In the  
 25 embodiment of Figure 7, from field 411 is represented in registers 144 of hardware processor 100. Mutator process 610, garbage collector process 620, and partially relocated object trap handler 650 include software executable on hardware processor 100. This embodiment facilitates incremental copying of large objects. However, without additional support, this embodiment is not well suited to incremental update of large numbers of pointers to the ToSpace 430 instance of a popular relocated object. Nonetheless, for garbage  
 30 collection systems in which objects are referenced via handles, for example, as described below with reference to Figure 17B, such an embodiment reduces hardware requirements and is viable for bounded pause time relocation since updating a single handle to the ToSpace 430 instance can be trivial.

Mutator process 610 makes read and write accesses from and to collected space 630. Read accesses are unaffected by garbage collection support. However, write accesses are selectively trapped by hardware  
 35 write barrier 740. Hardware write barrier 740 triggers partially relocated object trap handler 750 in response



- 15 -

to a correspondence between the contents of from field 411 and the target *objectref* associated with write access 701. In this embodiment, partially relocated object trap handler 750, determines, or optionally maintains memory storage for, the Copy-To destination information such as that otherwise stored in to field 412. In addition, garbage collection software may optionally maintain storage for HowFar information such as that otherwise stored in howfar field 413. When mutator process 610 stores to the FromSpace 420 instance of the large object, store data is either broadcast to both the copy-from instance 450 and copy-to instance 450A or, in embodiments which provide storage for howfar information, the howfar storage is inspected and the store is broadcast to both instances if the stored-to field of the large object has already been copied, and otherwise directed to copy-from instance 450. Those of skill in the art will appreciate a variety of methods by which garbage collector process 620 may update to field and howfar field storing variables 751 available to partially relocated object trap handler 750. Update 701 is by any such suitable method.

As described above, garbage collector process 620 updates from field 411 of partially relocated object identifier store 410 to identify copy-from instance 450 of the large object. In this embodiment, no pointers to the copy-to instance are available to mutator process 610 until the handle for the large object has been updated after copying is complete. Therefore, no barrier to accesses to copy-to instance 450A need be maintained. This embodiment is not configured for overlapped copy-from and copy-to instances; however, support for overlapped instances is not necessary for many relocating collector methods, e.g., generational scavenging or copying collector methods.

In one embodiment, hardware write barrier 740 is provided by integer unit 142 (Fig. 1) logic responsible for evaluating a store-oriented bytecode (e.g., *putfield\_quick*, *putfield*, *aastore*, etc.). In such an embodiment, logic implementing hardware write barrier 740 traps writes to the Copy-From instance identified by contents of from field 411. For example, the desired behavior of hardware write barrier 640A is described by the following logic equations.

```

    if (objectref == FROM) then
        generate gc_notify

```

where *gc\_notify* is an exemplary trap handler. Those of skill in the art will appreciate a variety of suitable logic implementations, including logic implementations combining other write barrier functionality such as intergenerational pointer store trapping, as described above. How far information may optionally be provided as howfar field 413 in registers 144 (see Fig. 4, not shown in Fig. 7) to better tailor trapping by hardware write barrier 740 to only those situations in which broadcast is necessary, thereby reducing overheads associated with partially relocated object trap handler 750 invocations.

Figure 8 depicts an embodiment including a Copy-To register field (e.g., to field 412) in addition to a Copy-From register field (e.g., from field 411) with associated write barrier 840. The embodiment of Figure 8 provides a write barrier to stores to fields of an object identified by the contents of from field 411 and

- 16 -

advantageously eliminates partially relocated object trap handler 750 and its associated overheads. This embodiment facilitates incremental (bounded pause time) copying of large objects, but like that of Figure 7, is not well suited to bounded pause time update of large numbers of pointers to the ToSpace 430 instance of a copied object. As before, handled object references can mitigate this limitation, allowing overall bounded  
 5 pause time relocation of even popular large objects, albeit at the expense of an additional level of indirection.

By maintaining a Copy-To register field in hardware, e.g., as to field 412 in registers 144, write accesses to the Copy-From instance can be broadcast to both the Copy-From and Copy-To instances by hardware write barrier 840 without software trap handler intervention. In one embodiment, hardware write barrier 840 is provided by integer unit 142 (Fig. 1) logic responsible for evaluating a store-oriented bytecode.  
 10 In such an embodiment, logic implementing hardware write barrier 840 broadcasts *store\_data* to both Copy-From and Copy-To instances respectively identified by *objectref* and contents of to field 412. An exemplary hardware write barrier 840 is described by the following logic equations.

```

    if (objectref == FROM) {
        store_data => *(objectref + offset)
  15      store_data => *(TO + offset)
    }
  
```

where *offset* is the offset into the large object of the target field associated with the store-oriented bytecode. Those of skill in the art will appreciate a variety of suitable logic implementations, including logic implementations combining other write barrier functionality such as intergenerational pointer store trapping,  
 20 as described above. Because to field 412 is available to hardware write barrier 840, the broadcast can be performed in hardware and software trap overhead is eliminated.

Figure 9 depicts an embodiment including Copy-To and How-Far register fields (e.g., to field 412 and howfar field 413) in addition to a Copy-From register field (e.g., from field 411) with associated hardware write barrier 940. The embodiment of Figure 9 provides a write barrier to stores to fields of an object  
 25 identified by the contents of from field 411 and advantageously allows hardware write barrier 940 to forgo broadcasting of stores to a not yet copied portion of the large object, while eliminating partially relocated object trap handler 650 and its associated overheads. Like the above embodiments, this embodiment facilitates incremental copying of large objects, but is not well suited to bounded pause time update of large numbers of pointers to the ToSpace 430 instance of a copied object. As before, handled object references can  
 30 mitigate this limitation, allowing overall bounded pause time relocation of even popular large objects, albeit at the expense of an additional level of indirection.

By maintaining a Copy-To register field in hardware, e.g., as to field 412 in registers 144, write accesses to the Copy-From instance can be broadcast to both the Copy-From and Copy-To instances by hardware write barrier 940. By further maintaining a How-Far register field in hardware, e.g., as howfar field

- 17 -

413 in registers 144, broadcasting can be limited to those write accesses for which the particular target field of the write access has already been copied to ToSpace 430. In either case, handling of the write access to the partially relocated object is without software trap handler intervention. In one embodiment, hardware write barrier 940 is provided by integer unit 142 (Fig. 1) logic responsible for evaluating a store-oriented bytecode.

5 In such an embodiment, logic implementing hardware write barrier 940 broadcasts `store_data` to both the Copy-From and the Copy-To instances if the target object field is in an already copied portion of the large object, and directs `store_data` to the Copy-From instance if the target object field is in a not yet copied portion of the large object. An exemplary hardware write barrier 940 is described by the following logic equations.

```

10      if (objectref == FROM) {
          if offset > HOWFAR then
              store_data => *(TO + offset)
              store_data => *(objectref + offset)
          }

```

15 where *offset* is the offset into the large object of the target field associated with the store-oriented bytecode. In some embodiments, the directing of `store_data` to the Copy-From instance may be provided by simply allowing the write access to complete normally without hardware write barrier 940 intervention. Those of skill in the art will appreciate a variety of suitable logic implementations, including logic implementations combining other write barrier functionality such as intergenerational pointer store trapping, as described

20 above. Because *howfar* field 413 is available to hardware write barrier 940, hardware write barrier 940 can selectively forgo writing `store_data` to the Copy-To instance. This slight optimization may improve performance by performing only a single write rather than broadcasting two writes.

*Write Barrier Associated with both Copy-From and Copy-To Instance Identifier Stores*

Additional embodiments are now described with reference to Figures 10 and 11. Like some of the

25 previously described embodiments, these additional embodiments exploit hardware support (e.g., a hardware write barrier and partially relocated object identifier store 410) to appropriately handle writes to the FromSpace 420 instance of a partially relocated large object without software trap handler intervention. However, in addition, these additional embodiments provide a barrier to writes to the ToSpace 430 instance of a partially relocated large object. In this way, these additional embodiments, support both incremental

30 copying of a large object and incremental update of pointers to a popular object for overall bounded pause time relocation of large and/or popular objects without handled object references.

In the embodiments of Figures 10 and 11, garbage collector process 620 incrementally updates pointers (i.e., *objectrefs*) to the large object. Mutator process 610 write accesses to either the FromSpace 420 instance or the ToSpace 430 instance are broadcast (or appropriately directed) by a hardware write barrier,

- 18 -

e.g., hardware write barrier 1040 or hardware write barrier 1140, so that the two instances are kept "in sync" while the *objectrefs* are updated. Because the two instances are kept in sync, the memory referencing structure is robust to referencing states in which some *objectrefs* refer to a FromSpace 420 instance and others refer to the corresponding ToSpace 430 instance. Bounded pause time update of *objectrefs* referring to the large  
 5 object is accomplished by incrementally updating such *objectrefs* to refer to the ToSpace 430 instance (rather than the FromSpace 420 instance). Note that in the embodiments of Figures 10 and 11, the FromSpace 420 instance and the ToSpace 430 instance may not overlap.

Referring to Figure 10, hardware write barrier 1040 provides a barrier to write accesses to either copy-from instance 450 or copy-to instance 450A. Hardware write barrier 1040 supports incremental,  
 10 bounded-pause time copying as described above with reference to hardware write barrier 840. However, hardware write barrier 1040 is also responsive to Copy-To register field (e.g., to field 412). Write accesses to either copy-from instance 450 or copy-to instance 450A are broadcast to both copy-from instance 450 and copy-to instance 450A so that the data states of the two instances are synchronized. In this way, *objectrefs* to the partially relocated large object may be updated incrementally, since a read access to either instance will  
 15 resolve to the same field state.

In one embodiment, hardware write barrier 1040 is provided by integer unit 142 (Fig. 1) logic responsible for evaluating a store-oriented bytecode. In such an embodiment, logic implementing hardware write barrier 1040 broadcasts *store\_data* to both Copy-From and Copy-To instances respectively identified by the contents of from field 411 and to field 412. An exemplary hardware write barrier 1040 is  
 20 described by the following logic equations.

```

    if ((objectref == FROM) || (objectref == TO)) {
        store_data => *(FROM + offset)
        store_data => *(TO + offset)
    }
  
```

25 where *offset* is the offset into the large object of the target field associated with the store-oriented bytecode. Those of skill in the art will appreciate a variety of suitable logic implementations, including logic implementations combining other write barrier functionality such as intergenerational pointer store trapping, as described above. Because from field 411 and to field 412 are available to hardware write barrier 1040, stores to the copy-from instance 450 or copy-to instance 450A respectively identified thereby can be  
 30 recognized and the broadcast to both instances can be performed in hardware without software trap overhead.

Figure 11 depicts an embodiment that adds a How-Far register field (e.g., *howfar* field 413) and a Mode indication (e.g., *mode* field 1114 of registers 144). As in the embodiment of Figure 10, hardware write barrier 1140 provides a barrier to write accesses to either copy-from instance 450 or copy-to instance 450A. Hardware write barrier 1140 supports incremental, bounded-pause time copying by ensuring that copied

- 19 -

portions of FromSpace 420 and ToSpace 430 instances are kept in sync. Hardware write barrier 1140 advantageously allows hardware processor 100 to forgo broadcasting of stores to a not yet copied portion of the large object during a copying phase, which in the embodiment of Figure 11, is indicated by a COPY state of mode field 1114. However, during a pointer update phase, indicated by a PTR\_UPDATE state of mode field 1114, writes accesses to either copy-from instance 450 or copy-to instance 450A are broadcast to both copy-from instance 450 and copy-to instance 450A. Garbage collector process 620 updates mode field 1114 to correspond to a current phase of a large object relocation.

In one embodiment, hardware write barrier 1140 is provided by integer unit 142 (Fig. 1) logic responsible for evaluating a store-oriented bytecode. In such an embodiment, logic implementing hardware write barrier 1140 broadcasts `store_data` to both the Copy-From and the Copy-To instances if the target object field is in an already copied portion of the large object or if the large object has been completely copied and relocation has entered a pointer update phase. During the copying phase, logic implementing hardware write barrier 1140 directs `store_data` to the Copy-From instance if the target object field is in a not yet copied portion of the large object. An exemplary hardware write barrier 1140 is described by the following logic equations.

```

10      if ((objectref == FROM) || (objectref == TO)) {
           if ((offset ≥ HOWFAR) || MODE == PTR_UPDATE)) then
               store_data => *(TO + offset)
               store_data => *(FROM + offset)
20      }

```

where the state of mode field 1114, MODE indicates a relocation phase (e.g., COPY or PTR\_UPDATE) and *offset* is the offset into the large object of the target field associated with the store-oriented bytecode. Those of skill in the art will appreciate a variety of suitable logic implementations, including logic implementations combining other write barrier functionality such as intergenerational pointer store trapping, as described above. Because howfar field 413 and mode field 1114 are available to hardware write barrier 1140, hardware write barrier 1140 can selectively forgo writing `store_data` to the Copy-To instance during the copying phase of a large object relocation, while broadcasting writes during the pointer update phase. This slight optimization may improve performance by performing only a single write, rather than broadcasting two writes, during the copying phase. Additionally, relocation mode may be indicated by allowing garbage collector process 620 to encode completion of the copying phase in a state of howfar field 413. Those of skill in the art will recognize that in the above logic equations, a howfar field 413 value of less than or equal to zero (0) may be used to encode completion of the copy phase thereby obviating mode field 1114.

- 20 -

Read and Write Barriers Associated with Copy-From Instance Identifier Store

Additional embodiments are now described with reference to Figures 12-14. Like some of the previously described embodiments, these additional embodiments exploit hardware support (e.g., a hardware write barrier and partially relocated object identifier store 410) to appropriately handle writes to the

5 FromSpace 420 instance of a partially relocated large object. However, these additional embodiments also provide a barrier to reads from a FromSpace 420 instance of a partially relocated large object. In this way, these additional embodiments support bounded-pause time copying as well as bounded pause time update of pointers to the large object without handled object references.

In the embodiments of Figures 12-14, garbage collector process 620 incrementally copies the large

10 object from a FromSpace 420 instance to a ToSpace 430 instance during a copy phase (MODE = COPY) and incrementally updates pointers (i.e., *objectrefs*) to the large object during a pointer update phase (MODE = PTR\_UPDATE). During the copy phase of large object relocation, mutator process 610 write accesses to the FromSpace 420 instance are broadcast or appropriately directed by a partially relocated object trap handler, e.g., partially relocated object trap handler 1250 (Fig. 12), or by a hardware write barrier, e.g.,

15 hardware write barrier 1340 (Fig. 13) or hardware write barrier 1440 (Fig. 14). During the pointer update phase of large object relocation, mutator process 610 write and read accesses to the FromSpace 420 instance are redirected to the ToSpace 430 instance.

Figure 12 depicts an embodiment including a Copy-From register field with associated read and write barriers. In particular, the embodiment of Figure 12 includes from field 411, hardware read barrier 1242,

20 hardware write barrier 1241, and partially relocated object trap handler 1250. Hardware read barrier 1242 and hardware write barrier 1241 are shown illustratively as read and write barrier hardware 1240. However, depending on the implementation, hardware write barrier 1241 and hardware read barrier 1242 may share hardware or be based on separate hardware. In one embodiment, integer unit 142 (Fig. 1) logic responsible for evaluating store-oriented bytecodes provides hardware write barrier 1241 and integer unit 142 logic

25 responsible for evaluating load-oriented bytecodes (e.g., *getfield\_quick*, *getfield*, *aaload*, etc.) provides hardware read barrier 1242.

Mutator process 610 makes read and write accesses from and to collected space 630. A given read or write access is selectively trapped by read and write barrier hardware 1240 if the read or write access targets the FromSpace 420 instance of a partially relocated object. Read and write barrier hardware 1240 triggers

30 partially relocated object trap handler 1250 in response to a correspondence between the contents of from field 411 and the target *objectref* associated with write access 701 or read access 1202. In one embodiment, read and write barrier hardware 1240 is responsive to a mode field 1114 of registers 144 which alters the behavior of hardware read barrier 1242 based on mode field 1114 state. In such an embodiment, hardware read barrier 1242 selectively traps only during the pointer update phase of large object relocation.

- 21 -

An exemplary hardware write barrier 1241 is described by the following logic equations.

```
if (objectref == FROM) then
    generate gc_notify
```

where gc\_notify is a trap handler such as partially relocated object trap handler 1250. Those of skill in the art will appreciate a variety of suitable logic implementations, including logic implementations combining other write barrier functionality such as intergenerational pointer store trapping, as described above. An exemplary hardware read barrier 1242 is described by the following logic equations.

```
if ((MODE == PTR_UPDATE) && (objectref == FROM)) then
    generate gc_notify
```

Alternative embodiments may selectively trap during both copying and pointer update phases of large object relocation with appropriate handling by partially relocated object trap handler 1250, though at the expense of greater trap handler overhead.

During the copying phase, partially relocated object trap handler 1250, determines, or optionally maintains memory storage for, the Copy-To destination information such as that otherwise stored in to field 412. In addition, partially relocated object trap handler 1250 may optionally maintain storage for HowFar information such as that otherwise stored in howfar field 413. When mutator process 610 stores to the FromSpace 420 instance of the large object, store data is either broadcast to both the copy-from instance 450 and copy-to instance 450A or, in embodiments which provide storage for howfar information, the howfar storage is inspected and the store is broadcast to both instances if the stored-to field of the large object has already been copied, and otherwise directed to copy-from instance 450.

During the pointer update phase, partially relocated object trap handler 1250 directs read and write accesses targeting a FromSpace 420 instance of a partially relocated large object to the ToSpace 430 instance thereof. As before, partially relocated object trap handler 1250, determines, or optionally maintains memory storage for, the Copy-To destination information such as that otherwise stored in to field 412. In either case, partially relocated object trap handler 1250 redirects both read and write accesses to the ToSpace 430 instance. In this way, a read access to the partially relocated object resolves to the ToSpace 430 instance which is guaranteed to be up to date.

Exemplary write trap and read trap functions for partially relocated object trap handler 1250 are as follows:

- 22 -

```

write_trap() {
    if (objectref == FROM) {
        store_data => *(TO + offset)
        if (MODE == COPY)
5         store_data => *(objectref + offset)
    }
}

read_trap() {
    if (objectref == FROM) {
10         if (MODE == PTR_UPDATE)
            read_destination <= *(TO + offset)
        else read_destination <= *(objectref + offset)
    }
}

```

15 where `read_destination` is the destination for a load-oriented bytecode.

In general, overhead imposed by trapping for the read-barrier case is likely to be greater than that in the case where only a write barrier is provided, because read accesses are typically more common than write accesses in a given instruction stream, and in any case, are in addition to write accesses. In part for this reason, hardware read barrier 1242 is selective for pointer update phase reads from FromSpace 420 instance of the partially relocated large object. However, hardware read barrier 1242 may optionally trap reads from the

20 FromSpace 420 instance of the partially relocated large object regardless of relocation phase. In such an embodiment, partially relocated object trap handler 1250 may be configured to intelligently handle an overlapping copy-from instance 450 and copy-to instance 450A. Figure 15 illustrates an overlapping from instance 1560 and to instance 1570 wherein a copied portion 1561 has been copied to portion 1571 of to

25 instance 1570. From field 411, to field 412, and howfar field 413 encode the partially-relocated object state as described above. In the embodiment of Figure 15, garbage collector process 620 copies the large object from back to front (or front to back) depending on the relative overlap of from instance 1560 and to instance 1570. To facilitate overlapped from and to instances, partially relocated object trap handler 1550 does not broadcast writes, and instead directs a write targeting not copied portion 1562 to from instance 1560 and directs a write

30 targeting copied portion 1561 to to instance 1570. A suitable modification to the write trap function of partially relocated object trap handler 1250 is as follows:



- 23 -

```

write_trap() {
    if (objectref == FROM) {
        if (COPY_DIRECTION == BACK_TO_FRONT) {
            if ((offset ≥ HOW_FAR) ||
5              (MODE == PTR_UPDATE))
                store_data => *(TO + offset)
            else store_data => *(objectref + offset)
        }
        else { /* COPY_DIRECTION == FRONT_TO_BACK */
10          if ((offset ≤ HOW_FAR) ||
              (MODE == PTR_UPDATE))
                store_data => *(TO + offset)
            else store_data => *(objectref + offset)
        }
15    }
}

```

where COPY\_DIRECTION is ascertainable from the relative positions of the from instance 1560 and to instance 1570 as respectively encoded by from field 411 and to field 412. Modifications to the read trap function of partially relocated object trap handler 1250 to appropriately redirect load-oriented accesses are analogous.

Figure 13 depicts an embodiment including a Copy-To register field (e.g., to field 412) in addition to a Copy-From register field (e.g., from field 411) with associated read and write barrier hardware 1340. As before, this embodiment includes a mode indication (e.g., mode field 1114 of registers 144) maintained by garbage collector process 620 to allow read and write barrier hardware 1340 to distinguish between a copy phase and a pointer update phase of large object relocation. Compared with the embodiment of Figure 12, that of Figure 13 provides both a read barrier and write barrier support in hardware (e.g., as hardware read barrier 1342 to stores to fields of an object identified by the contents of from field 411 and a hardware write barrier 1341 to stores to fields of an object identified by the contents of from field 411), thereby advantageously eliminating partially relocated object trap handler 1250 and its associated overheads.

By maintaining a Copy-To register field in hardware, e.g., as to field 412 in registers 144, write accesses to the Copy-From instance can be broadcast during a copy phase (MODE = COPY) to both the Copy-From and Copy-To instances by hardware write barrier 1341 without software trap handler intervention. During a pointer update phase (MODE = PTR\_UPDATE), write accesses to the Copy-From instance are redirected to the Copy-To instance by hardware write barrier 1341, again without software trap handler

- 24 -

intervention. Read accesses from the Copy-From instance are also redirected during the pointer update phase, to the Copy-To instance by hardware read barrier 1342 without software trap handler intervention. Copy phase read accesses to the Copy-From instance complete normally.

In one embodiment, read and write barrier hardware 1340 is provided by integer unit 142 (Fig. 1) logic responsible for evaluating bytecodes. Hardware write barrier 1341 is provided by integer unit 142 logic responsible for evaluating store-oriented bytecodes and hardware read barrier 1342 is provided by integer unit 142 logic responsible for evaluating load-oriented bytecodes.

During the copy phase, logic implementing hardware write barrier 1342 broadcasts *store\_data* to both Copy-From and Copy-To instances respectively identified by *objectref* and contents of field 412, while, during the pointer update phase logic implementing hardware write barrier 1342 redirects *store\_data* to only the Copy-To instance. An exemplary hardware write barrier 1342 is described by the following logic equations.

```

        if (objectref == FROM) {
            store_data => *(TO + offset)
        15         if (MODE == COPY)
                    store_data => *(objectref + offset)
        }

```

Similarly, an exemplary hardware read barrier 1341 is described by the following logic equations.

```

        if (objectref == FROM) {
        20         if (MODE == PTR_UPDATE)
                    read_destination <= *(TO + offset)
                    else read_destination <= *(objectref + offset)
        }

```

where, in each case, *offset* is the offset into the large object of the target field associated with the store-, or load-oriented bytecode. Those of skill in the art will appreciate a variety of suitable logic implementations, including logic implementations combining hardware read barrier 1341 and hardware write barrier 1342, as well as implementations combining hardware write barrier 1342 with other write barrier functionality such as intergenerational pointer store trapping, as described above. Because field 412 is available to hardware write barrier 1342 and to hardware read barrier 1341, broadcast and redirection can be performed in hardware and software trap overhead is eliminated.

Figure 14 depicts an embodiment that adds a How-Far register field (e.g., *howfar* field 413). As in the embodiment of Figure 13, read and write barrier hardware 1440 provides barriers to read and write

- 25 -

accesses from and to copy-from instance 450. In addition, read and write barrier hardware 1440 steers read and write accesses to an appropriate FromSpace 420 or ToSpace 430 instance depending on whether the field referenced by a store- or load-oriented bytecode has already been copied to the ToSpace 430 instance. Operating in conjunction with howfar field 413, hardware write barrier 1441 advantageously allows hardware processor 100 to forgo broadcasting of stores to a not yet copied portion of the large object during a copying phase indicated by a COPY state of mode field 1114. During the pointer update phase, ToSpace 430 instance is guaranteed to be up to date; therefore, by redirecting read accesses thereto, hardware read barrier 1442 allows hardware processor 100 to advantageously forgo broadcasting of stores during the pointer update phase. Because broadcast stores are eliminated, the embodiment of Figure 14 is particularly suited to support of overlapping FromSpace 420 and ToSpace 430 instances.

In one embodiment, read and write barrier hardware 1440 is provided by integer unit 142 (Fig. 1) logic responsible for evaluating bytecodes. Hardware write barrier 1441 is provided by integer unit 142 logic responsible for evaluating store-oriented bytecodes and hardware read barrier 1442 is provided by integer unit 142 logic responsible for evaluating load-oriented bytecodes. An exemplary hardware write barrier 1442 is described by the following logic equations.

```

    if (objectref == FROM) {
        if (COPY_DIRECTION == BACK_TO_FRONT) {
            if ((offset ≥ HOW_FAR) ||
                (MODE == PTR_UPDATE))
                store_data => *(TO + offset)
            else store_data => *(objectref + offset)
        }
        else { /* COPY_DIRECTION == FRONT_TO_BACK */
            if ((offset ≤ HOW_FAR) ||
                (MODE == PTR_UPDATE))
                store_data => *(TO + offset)
            else store_data => *(objectref + offset)
        }
    }

```

Similarly, an exemplary hardware read barrier 1341 is described by the following logic equations.

- 26 -

```

    if (objectref == FROM) {
        if (COPY_DIRECTION == BACK_TO_FRONT) {
            if ((offset ≥ HOW_FAR) ||
                (MODE == PTR_UPDATE))
5               read_destination <= *(TO + offset)
            else read_destination <= *(objectref + offset)
        }
        else { /* COPY_DIRECTION == FRONT_TO_BACK */
            if ((offset ≤ HOW_FAR) ||
10              (MODE == PTR_UPDATE))
                read_destination <= *(TO + offset)
            else read_destination <= *(objectref + offset)
        }
    }
}

```

15 where `read_destination` is the destination for a load-oriented bytecode and where, in each case, *offset* is the offset into the large object of the target field associated with the store-, or load-oriented bytecode. Those of skill in the art will appreciate a variety of suitable logic implementations, including logic implementations combining hardware read barrier 1441 and hardware write barrier 1442, as well as implementations combining hardware write barrier 1442 with other write barrier functionality such as

20 intergenerational pointer store trapping, as described above.

*Read and Write Barriers Associated with Copy-To Instance Identifier Store*

A series of variations on the embodiments of Figures 12-14, include read and write barriers associated with a Copy-To instance identifier store, rather than with a Copy-From instance identifier store. As before, these variations add successive levels of additional hardware support, e.g., from field 411 and

25 howfar field 413 support, to eliminate software trap handler overhead and allow for overlapping copy-from and copy-to instances 450 and 450A. The variations support both bounded-pause time copying and bounded pause time update of pointers to a partially relocated large object. These variations are now described by analogy to the embodiments of Figures 12-14 and those of skill in the art will appreciate suitable modifications based on this description.

30 Garbage collector process 620 updates pointers to the large object, i.e., from the FromSpace 420 instance thereof to a ToSpace 430 instance thereof, before copying of the object data. In this way, the ordering of pointer update and copying phases is reversed. Read and write barriers forward read and write accesses directed at copy-to instance 450A instead to copy-from instance 450. In one embodiment, hardware read and write barriers trap accesses to copy-to instance 450A, invoking a software partially relocated object

- 27 -

trap handler that performs the actual forwarding to the Copy-From instance. The partially relocated object trap handler may optionally maintain storage for HowFar information such as that otherwise stored in howfar field 413. When mutator process 610 stores to the ToSpace 420 instance of the large object, store data is directed to the copy-to instance 450A if the stored-to field of the large object has already been copied, and otherwise directed to copy-from instance 450. Similarly, when mutator process 610 loads from the ToSpace 420 instance of the large object, the load is performed from the copy-to instance 450A if the stored-to field of the large object has already been copied, and otherwise performed from copy-from instance 450. Support for overlapped FromSpace 430 and ToSpace 430 instances is provided as described above with respect to the embodiments of Figures 12 and 14. Alternatively, if no support for overlapping regions is needed, the read barrier can steer loads to the Copy-From instance, and the write barrier can broadcast stores to both the Copy-From and Copy-To instances.

As above, software trap overhead can be eliminated by putting more registers and more complex behavior into the hardware. For example, one embodiment includes to field 412 and from field 411. A hardware write barrier responsive to correspondence between a store target and to field 412 broadcasts stores to both copy-to instance 450A and copy-from instance 450. A hardware read barrier responsive to correspondence between a load source and to field 412 redirects loads to copy-from instance 450. Another embodiment, includes howfar field 413 in addition to from field 411 and to field 412. The hardware write barrier and the hardware read barrier direct stores and loads to copied portions of copy-to instance 450A and to uncopied portions of copy-from instance 450. Those of skill in the art will appreciate suitable logic for each of the embodiments based on the foregoing description.

#### Object Referencing Formats

Figure 16 depicts one embodiment of an object reference (*objectref*) as represented in hardware processor 100. Three bits of the *objectref* can be used for garbage collection hints as described in the above-incorporated PCT International Application No. PCT/US98/07622, entitled, "GENERATION ISOLATION SYSTEM AND METHOD FOR GARBAGE COLLECTION." An additional handle bit H indicates whether the object is referenced by the *objectref* directly or indirectly-through a handle. Handles provide a referencing method that facilitates, albeit at the cost of an additional level of indirection, relocation of memory objects without large-scale updates of pointers (or *objectrefs*) thereto. Both of these fields are masked out before being provided to integer unit 142 (Fig. 1) of hardware processor 100.

In one embodiment of hardware processor 100 and collected space 630 (Figs. 6-14), an object 1700 is represented in memory including a header portion 1710 and an instance variable storage portion 1720. Figure 17A depicts one such embodiment. Header portion 1710 includes a 32-bit word that itself includes a method vector table base portion 1712 for representing object's class and five bits of additional storage 1714 reserved for synchronization status of the object and information for the garbage collector. Optionally, a second header-word, e.g., monitor pointer 1716, can contain the address of a monitor allocated for the object,

- 28 -

thereby making all five bits of additional storage 1714 in the first header word available for garbage collection information. In the embodiment of Figure 17A, an object reference (*objectref*) points to the location of method vector table base portion 1712 to minimize the overhead of method invocation.

Three bits of header portion 1710 are available to a garbage collector such as collector process 620. In header portion 1710, three lower-order-bits (header[2:0]), and two high-order-bits (header[31:30]) are masked off when the header is treated as a pointer. Three of these bits (header[31:30, 2]) are available to the garbage collector to store information about object 1700. Bits 1 and 0 may be used to hold LOCK and WANT bits for object synchronization. Alternatively, a second header word, e.g., monitor pointer 1716, can be provided for maintaining the synchronization status of object 1700, leaving all five bits for garbage collection support. How the bits for garbage collection support are used depends on the particular type(s) of garbage collection methods implemented by collector process 620 and garbage collection trap handler, if any. Possible uses include mark bits, counter bits to age objects within a generation, etc. As described above, in an optional second header-word embodiment of header portion 1710, five bits are available to a garbage collector such as collector process 620.

In the embodiment of Figure 17A, instance variable storage portion 1720 begins one word after the method vector table base portion 1712 and contains instance variables of object 1700. The least significant bit of an *objectref* specifies whether the reference is a handled (=1) or not (=0). An alternative, "handled," object format is depicted in Figure 17B. A handled reference is established when object 1700b is created and all subsequent references go through the handle, i.e., storage pointer 1750b to access the object. This support is provided for some types of garbage collectors which reduce costs of pointer updating during object relocation by copying handles rather than the underlying object storage, including that for instance variables.

Handled object references allow garbage collection systems, such as those described above with reference to Figs. 7-9, to exhibit bounded pause time performance for pointer updates to a copied object. In other garbage collection systems for which bounded pause time pointer update of large numbers of referencing pointers is supported by the barrier configuration provided thereby, for example, in the embodiments described above with reference to Figs. 10-14, direct object references are preferable.

While the invention has been described with reference to various embodiments, it will be understood that these embodiments are illustrative and that the scope of the invention is not limited to them. Claim terms such as first instruction, second instruction, third instruction, etc. are for identification only and should not be construed to require a particular ordering. Many variations, modifications, additions, and improvements of the embodiments described are possible. For example, although the present invention has been herein described with reference to exemplary embodiments relating to the JAVA programming language and JAVA virtual machine, it is not limited to them and, instead, encompasses systems, articles, methods, and apparatus for a wide variety of processor environments (both virtual and physical). In addition, although certain exemplary embodiments have been described in terms of hardware, suitable virtual machine implementations (JAVA related or otherwise) incorporating a partially relocated object identifier store and barrier(s) responsive thereto

- 29 -

in accordance with the above description include software virtual machine instruction processor implementations such as a bytecode interpreter or a just-in-time (JIT) compiler. These and other variations, modifications, additions, and improvements may fall within the scope of the invention as defined by the claims which follow.

**WHAT IS CLAIMED IS:**

- 1           1.       An apparatus comprising:  
2           memory storage, wherein objects formed therein are relocatable from respective FromSpace instances  
3           to respective ToSpace instances thereof;  
4           a partially relocated object identifier store updatable to identify a FromSpace instance of a particular  
5           one of said objects, if any, for which relocation is incomplete; and  
6           a write barrier to a store-oriented memory access targeting said FromSpace instance of said particular  
7           object, wherein said write barrier maintains consistency between said ToSpace instance and  
8           at least a copied portion of said FromSpace instance.
  
- 1           2.       An apparatus, as recited in claim 1, wherein said write barrier is responsive to a  
2           correspondence between contents of said partially relocated object identifier store and an object identifier for  
3           said store-oriented memory access.
  
- 1           3.       An apparatus, as recited in claim 1,  
2           wherein said write barrier comprises write barrier logic responsive to a correspondence between  
3           contents of said partially relocated object identifier store and an object identifier for said  
4           store-oriented memory access; and  
5           wherein said write barrier logic traps to a partially relocated object trap handler upon said  
6           correspondence.
  
- 1           4.       An apparatus, as recited in claim 3, further comprising:  
2           a ToSpace instance identifier accessible to said partially relocated object trap handler, said partially  
3           relocated object trap handler broadcasting a trapping store-oriented access to said ToSpace  
4           and said FromSpace instances.
  
- 1           5.       An apparatus, as recited in claim 4, wherein said partially relocated object identifier store  
2           includes said ToSpace instance identifier as a TO field accessible to said partially relocated object trap  
3           handler.
  
- 1           6.       An apparatus, as recited in claim 4, wherein said ToSpace instance identifier includes a  
2           portion of said memory storage accessible to said partially relocated object trap handler and to a garbage  
3           collector process.
  
- 1           7.       An apparatus, as recited in claim 3,  
2           further comprising a partial copy position indicator accessible to said partially relocated object trap  
3           handler,



- 31 -

4 wherein, for a trapping store-oriented access to a copied portion, said partially relocated object trap  
5 handler broadcasts to both said ToSpace and said FromSpace instances, and  
6 wherein, for a trapping store-oriented access to an uncopied portion of said particular object, said  
7 partially relocated object trap handler directs said store-oriented access to said FromSpace  
8 instance and not to said ToSpace instance.

1 8. An apparatus, as recited in claim 3,  
2 wherein said partially-relocated object identifier store includes a FromSpace instance identifier and a  
3 partial copy position indicator to identify a boundary between a copied portion and an  
4 uncopied portion of said particular object; and  
5 wherein said write barrier is responsive to said partially relocated object identifier store, selectively  
6 trapping to said partially relocated object trap handler upon a write to said copied portion of  
7 said particular object.

1 9. An apparatus, as recited in claim 1,  
2 wherein said partially relocated object identifier store includes FromSpace and ToSpace instance  
3 identifiers; and  
4 wherein said write barrier comprises write barrier logic coupled to said FromSpace and said ToSpace  
5 instance identifiers, said write barrier logic responsive to a correspondence between contents  
6 of said FromSpace instance identifier and an object identifier for a store-oriented memory  
7 access, and  
8 wherein, in response to said correspondence, said write barrier logic broadcasts said store-oriented  
9 access to said ToSpace and said FromSpace instances.

1 10. An apparatus, as recited in claim 1,  
2 wherein said write barrier comprises write barrier logic;  
3 wherein said partially relocated object identifier store comprises FromSpace and ToSpace instance  
4 identifiers and a partial copy position indicator, each coupled to said write barrier;  
5 wherein, for a store-oriented access to a copied portion of said particular object, said write barrier  
6 broadcasts store data to both said ToSpace and said FromSpace instances, and  
7 wherein, for a store-oriented access to an uncopied portion of said particular object, said write barrier  
8 directs said store data to said FromSpace instance and not to said ToSpace instance.

1 11. An apparatus, as recited in claim 1, further comprising:  
2 a garbage collection process including first instructions executable on a processor including said  
3 partially relocated object identifier store and said write barrier, said first instructions for  
4 incrementally copying portions of said particular object from said FromSpace instance to  
5 said ToSpace instance; and

- 32 -

6 a mutator process including second instructions executable on said processor for making load-  
7 oriented memory accesses and said store-oriented memory access to said particular object.

1 12. An apparatus, as recited in claim 1, wherein said partially relocated object identifier store  
2 includes a partial copy position indicator portion indicative of a boundary between a copied portion and an  
3 uncopied portion of said particular object.

1 13. An apparatus, as recited in claim 1, wherein said relocation comprises incrementally copying  
2 said particular object from a FromSpace portion to a ToSpace portion of a garbage collected portion of said  
3 memory storage.

1 14. An apparatus, as recited in claim 1,  
2 wherein said FromSpace and said ToSpace overlap; and  
3 wherein said relocation comprises incrementally copying said particular object to compact a plurality  
4 of said objects in said memory storage.

1 15. An apparatus, as recited in claim 1, wherein said partially relocated object identifier store is  
2 updated by a garbage collection process in correspondence with incremental copying thereby.

1 16. An apparatus, as recited in claim 1, further comprising:  
2 a processor embodying said partially relocated object identifier store and said write barrier; and  
3 mutator process instructions encoded in media readable by said processor and executable thereby,  
4 said mutator process instructions including an instruction corresponding to said store-  
5 oriented memory access, and  
6 relocater process instructions encoded in media readable by said processor and executable thereby,  
7 said garbage collector process instructions including an instruction sequence to  
8 incrementally copy said particular object from said FromSpace instance to said ToSpace  
9 instance, to update an object reference handle to reference said ToSpace instance, and to  
10 maintain said partially relocated object identifier store in correspondence therewith, said  
11 instruction sequence being interruptable by said mutator process instructions.

1 17. An apparatus, as recited in claim 16, wherein said relocater process instructions are selected  
2 to perform one of generational garbage collection, mark-sweep-compact garbage collection, copying garbage  
3 collection, and heap compaction with bounded pause time.

1 18. An apparatus, as recited in claim 16, wherein said processor comprises a hardware processor  
2 including register storage and logic embodying said partially relocated object identifier store and said write  
3 barrier.

- 33 -

1           19.     An apparatus, as recited in claim 1, further comprising:  
2           a processor embodying said partially relocated object identifier store and said write barrier; and  
3           mutator process instructions encoded in media readable by said processor and executable thereby,  
4           said mutator process instructions including an instruction corresponding to said store-  
5           oriented memory access, and  
6           relocator process instructions encoded in media readable by said processor and executable thereby,  
7           said garbage collector process instructions including an instruction sequence to  
8           incrementally copy said particular object from said FromSpace instance to said ToSpace  
9           instance, to update pointers referencing said FromSpace instance to reference said ToSpace  
10          instance, and to maintain said partially relocated object identifier store in correspondence  
11          therewith, said instruction sequence being interruptable by said mutator process instructions.

1           20.     An apparatus, as recited in claim 1, wherein said particular object comprises one of a large  
2           object, a popular object, and a large and popular object.

1           21.     An apparatus, as recited in claim 1,  
2           a hardware processor embodying said partially relocated object identifier store and said write barrier;  
3           a communications device coupled to said hardware processor for receiving mutator process  
4           instructions for execution on said hardware processor, said mutator process instructions  
5           including a first instruction corresponding to said store-oriented memory access; and  
6           media readable by said hardware processor to encode garbage collector process instructions  
7           executable and said hardware processor for incrementally copying said particular object.

1           22.     An apparatus, as recited in claim 1,  
2           further comprising first and second processors coupled to said partially relocated object identifier  
3           store and to said memory storage, said first processor including said write barrier;  
4           wherein said relocation comprises incremental copying of said particular object by a garbage  
5           collector process executable on said second processor, and  
6           wherein said store-oriented memory access comprises a write access by a mutator process executable  
7           on said first processor; and  
8           wherein said partially relocated object identifier store is maintained by said garbage collector process  
9           in correspondence with said incremental copying of said particular object.

1           23.     An apparatus, as recited in claim 22, wherein said second processor is one of a special  
2           purpose garbage collection processor; integral with said memory storage, said second  
3           processor and said memory storage together comprising a garbage collected memory module  
4           and a virtual machine instruction processor.

- 34 -

1       24.     An apparatus, as recited in claim 1,  
2       wherein said partially relocated object identifier store is further updatable to identify a second of said  
3               objects for which relocation is incomplete; and  
4       wherein said apparatus further comprises first and second garbage collection processors coupled to  
5               said partially relocated object identifier store and to said memory storage, relocation of said  
6               particular and said second objects comprising incremental copying thereof by said first and  
7               said second garbage collection processors, respectively.

1       25.     A garbage collection system to manage memory object instance consistency during  
2       relocation of a particular memory object in a computer system providing bounded pause-time relocation of  
3       said particular memory object using incremental copying of said particular object from a FromSpace instance  
4       to a ToSpace instance thereof, said garbage collection system comprising:  
5               a partially relocated object identifier store including identifier fields for said FromSpace and said  
6               ToSpace instances;  
7       write barrier logic responsive to a correspondence between a target object identifier for a store-  
8               oriented mutator process access to said particular object and contents of said FromSpace  
9               instance identifier field, wherein said write barrier maintains consistency between said  
10              FromSpace and said ToSpace instances during said incremental copying.

1       26.     A garbage collection system, as recited in claim 27,  
2       wherein said consistency maintenance includes broadcasting a first store-oriented mutator process  
3               access targeting said FromSpace instance of said particular object to both said FromSpace  
4               and said ToSpace instances thereof.

1       27.     A garbage collection system, as recited in claim 27,  
2       wherein said partially relocated object identifier store further includes a partial copy position  
3               indicator to identify an uncopied portion of said particular object; and  
4       wherein said write barrier logic selectively forgoes broadcast for a second store-oriented mutator  
5               process access to said uncopied portion, and instead directs such access to said FromSpace  
6               instance.

1       28.     A method for bounding garbage collection pause time, said method comprising:  
2       during relocating of a memory object from a source instance to a target instance thereof, interrupting  
3               said relocating in accordance with a bounded pause time commitment to a mutator process;  
4       selectively trapping, during said interruption of said relocating, a write access by said mutator process  
5               to said memory object by detecting a correspondence between an object identifier therefor  
6               and contents of a partially relocated object identifier store; and

- 35 -

7 in response to said trapping, handling said write access in accordance with a partial relocation state of  
8 said memory object.

1 29. A method, as recited in claim 30,  
2 wherein, if there exists a correspondence between a copied portion of said memory object and said  
3 write access, said handling includes broadcasting said write access to both of said source  
4 instance and said target instance; and  
5 otherwise, completing said write access without broadcasting.

1 30. A method, as recited in claim 30,  
2 wherein said selective trapping includes further detecting a correspondence between a copied portion  
3 of said memory object and said write access such that a particular write access to an  
4 uncopied portion of said memory object is untrapped.

1 31. A method for relocating a memory object with bounded pause time impact on a mutator  
2 process having access thereto, said method comprising:  
3 configuring a write barrier to respond to store-oriented accesses of said mutator process targeting a  
4 From instance of said memory object;  
5 incrementally copying said memory object from said From instance to said To instance;  
6 during said incremental copying and in response to a first of said store-oriented accesses,  
7 broadcasting said first store-oriented access to both said From and said To instances.

1 32. A method, as recited in claim 33,  
2 wherein said write barrier configuring comprises updating a partially relocated object identifier store  
3 to identify said From instance; and  
4 wherein said broadcasting is in response to a correspondence between an object identifier for said  
5 first store-oriented access and contents of said partially relocated object identifier store.

1 33. A method, as recited in claim 34, wherein said write barrier configuring further comprises  
2 updating a partially relocated object identifier store to identify said To instance.

1 34. A method, as recited in claim 33, further comprising:  
2 maintaining a partial copy position indicator to discriminate between a copied portion and an  
3 uncopied portion of said memory object; and  
4 during said incremental copying and in response to a second of said store-oriented accesses targeting  
5 said uncopied portion, directing said second store-oriented access to said From instance and  
6 not to said To instance.

- 36 -

1           35.     A method, as recited in claim 33, wherein said broadcasting is performed by one of said  
2     write barrier and trap handler software in response to a trap by said write barrier.

1           36.     Support for a bounded pause time implementation of a relocating garbage collector in a  
2     virtual machine instruction processor, said support comprising:  
3         a partially relocated memory object identifier store; and  
4         a write barrier responsive to said partially relocated memory object identifier store such that  
5             evaluation of a store instruction to a memory object identified in said partially relocated  
6             memory object identifier store traps.

1           37.     Support for a bounded pause time implementation of a relocating garbage collector in a  
2     virtual machine instruction processor, said support further comprising:  
3         a trap handler responsive to said write barrier.

1           38.     A computing machine comprising:  
2         a processor;  
3         memory, wherein objects formed therein are referenceable by said processor; and  
4         means accessible by said processor for identifying a partially collected object in said memory,  
5             wherein a correspondence between said partially collected object identifier means and an  
6             object identifier for an object reference operation by said processor triggers fault handler  
7             means;  
8         means for garbage collecting said memory, wherein said garbage collecting means maintains said  
9             partially collected object identifier means to identify a particular one, if any, of said objects  
10            for which garbage collection is incomplete, thereby allowing interruption of said garbage  
11            collecting means in accordance with a bounded pause time commitment to mutator means  
12            executable on said processor.

1           39.     A computer program product comprising:  
2         an interruptable object relocater encoded in media readable by a processor and executable on said  
3         processor, wherein said interruptable object relocater causes, at least upon interruption, a  
4         partially relocated object identifier store of said processor to identify a from instance of a  
5         partially relocated memory object;  
6         a partially relocated object access handler invoked in response to a correspondence between contents  
7             of said partially relocated object identifier store and an object identifier for an object  
8             reference operation by a mutator executable on said processor, wherein said partially  
9             relocated object access handler handles said object reference operation in accordance with a  
10            partial relocation state of said partially relocated memory object.

- 37 -

1           40.     A computer program product, as recited in claim 41, wherein said interruptable object  
2     relocator provides object copying from a FromSpace portion of memory to a ToSpace portion of memory in  
3     accordance with a copying collector.

1           41.     A computer program product, as recited in claim 41, wherein said interruptable object  
2     relocator provides object copying from a source instance to a potentially overlapping target instance within a  
3     collected portion of memory in accordance with a compacting operation.

1           42.     A computer program product, as recited in claim 41, wherein said interruptable object  
2     relocator provides object relocation within a collected portion of memory in accordance with a mark-compact  
3     collector implementation.

1           43.     A computer program product, as recited in claim 41, wherein said interruptable object  
2     relocator provides object copying from a young-portion of memory to a older-portion of memory in  
3     accordance with a generational collector implementation.

1/17

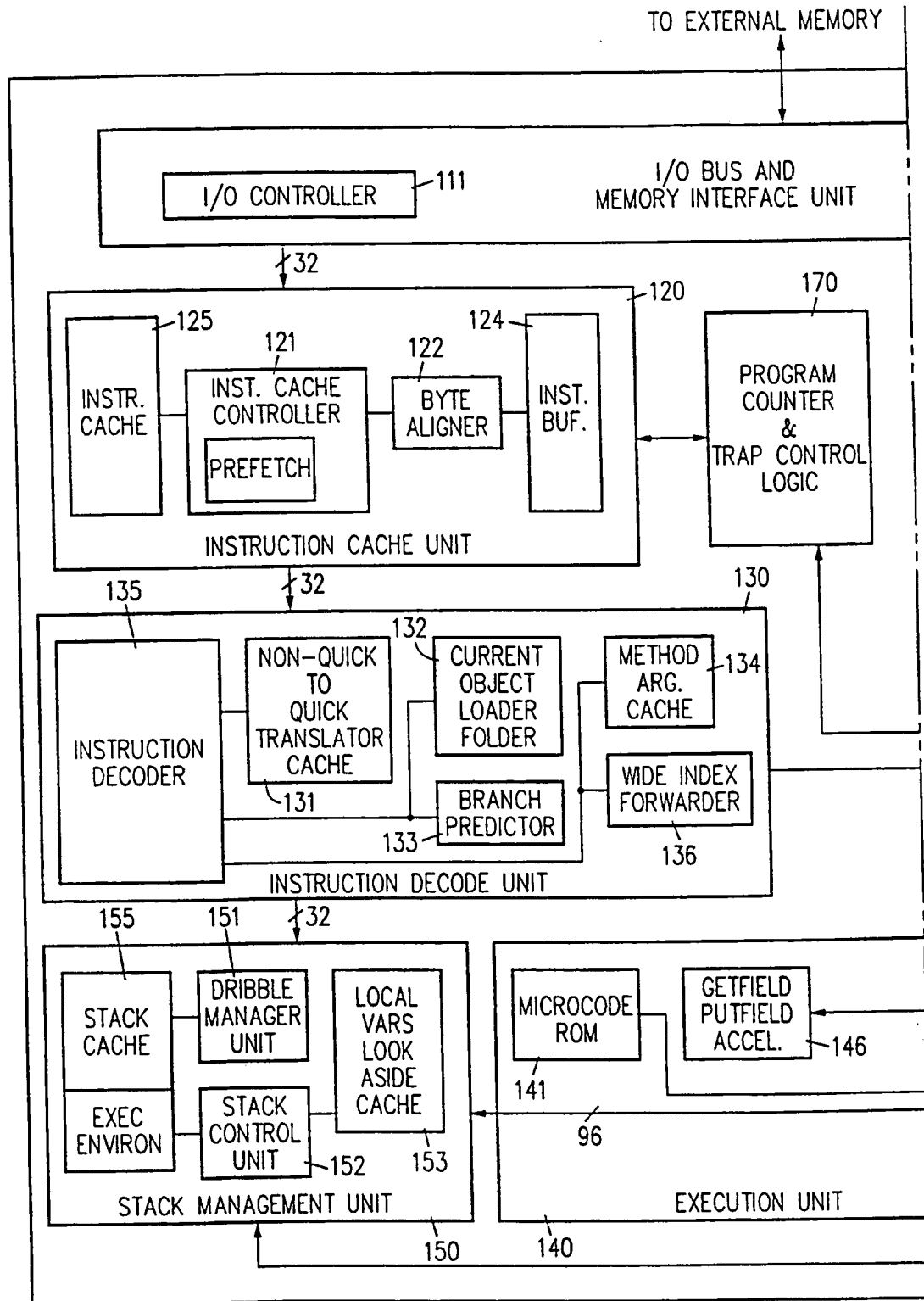


FIG. 1A



2/17

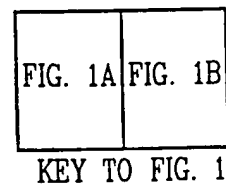
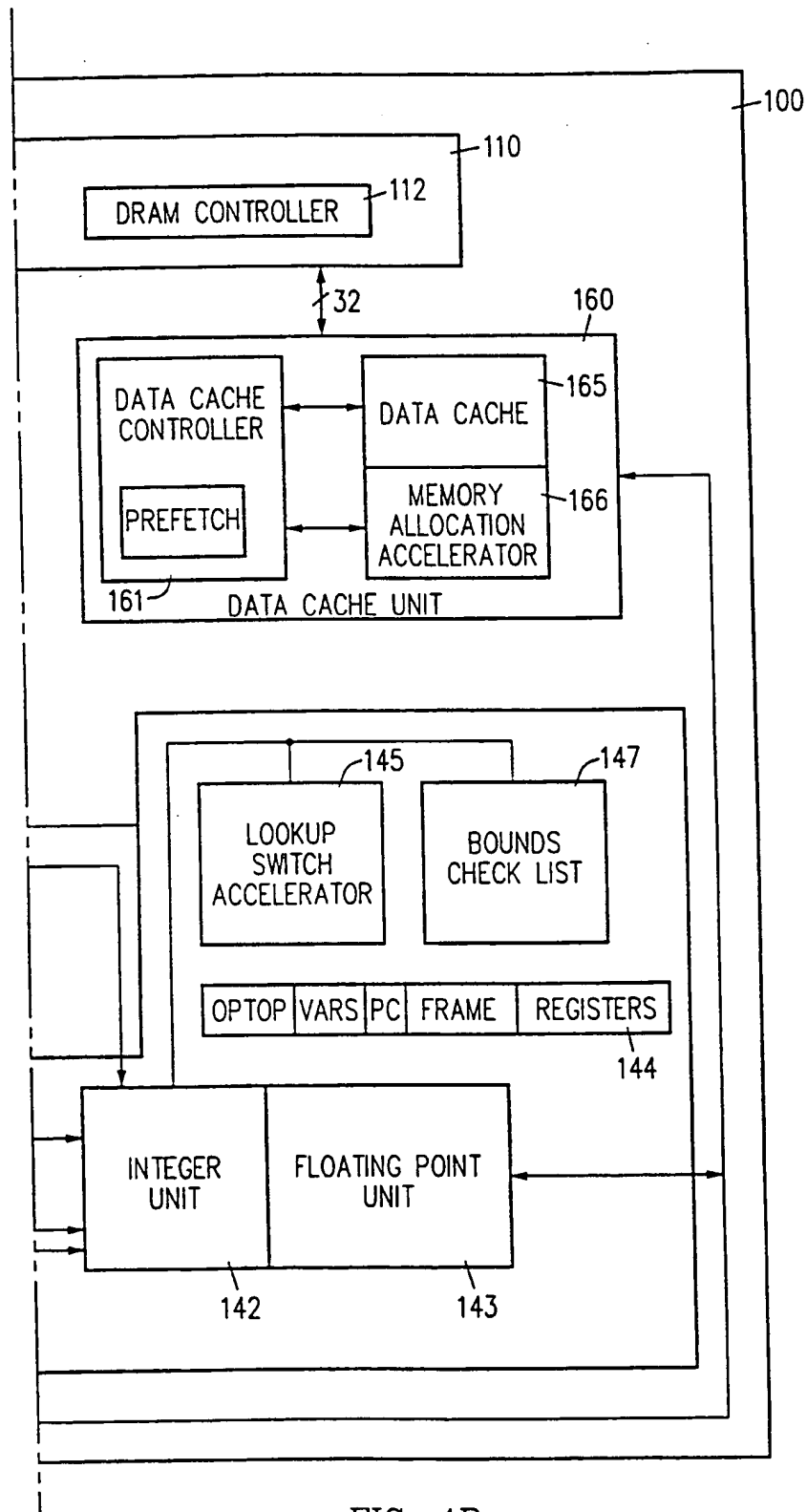
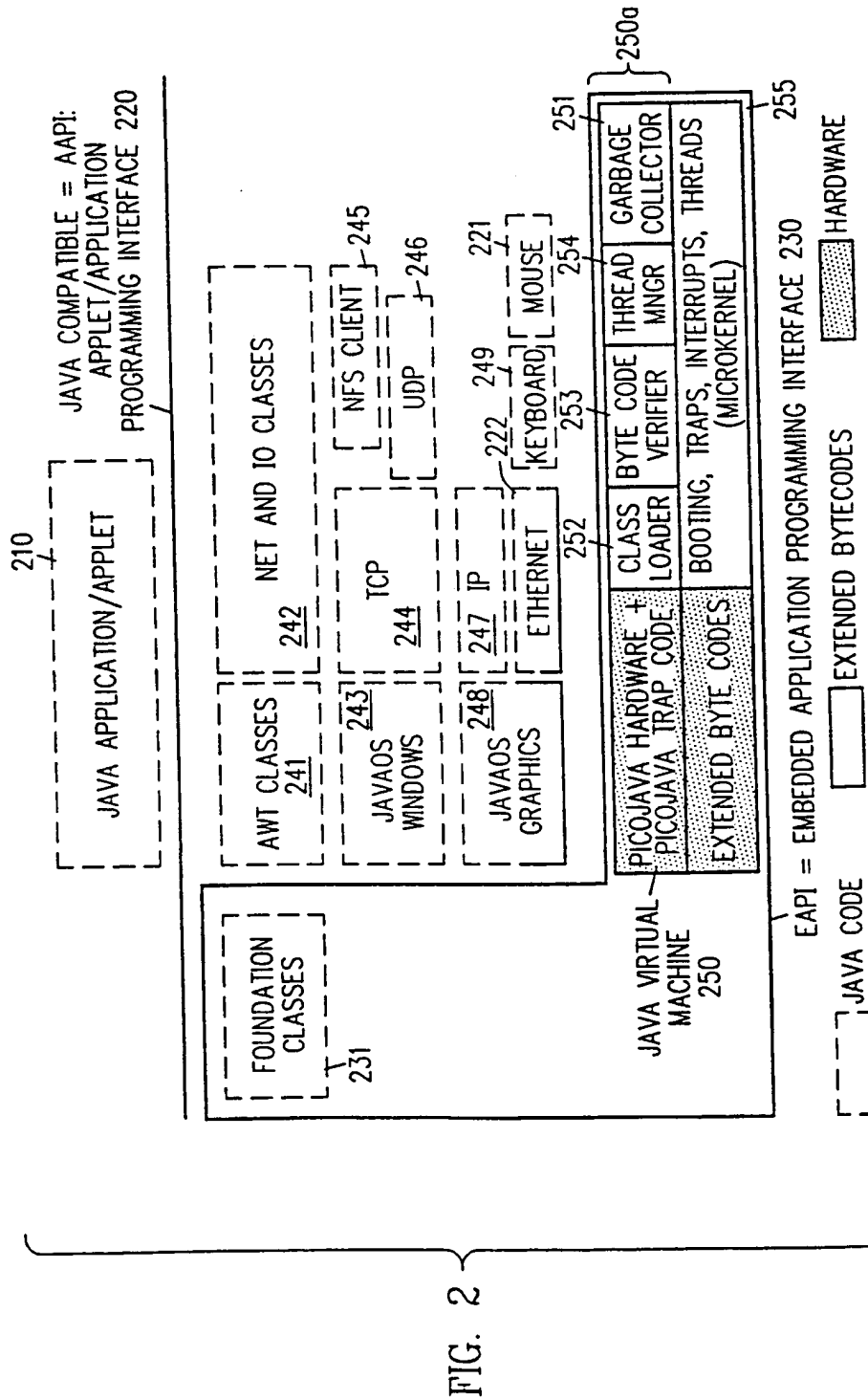


FIG. 1B

3/17



4/17

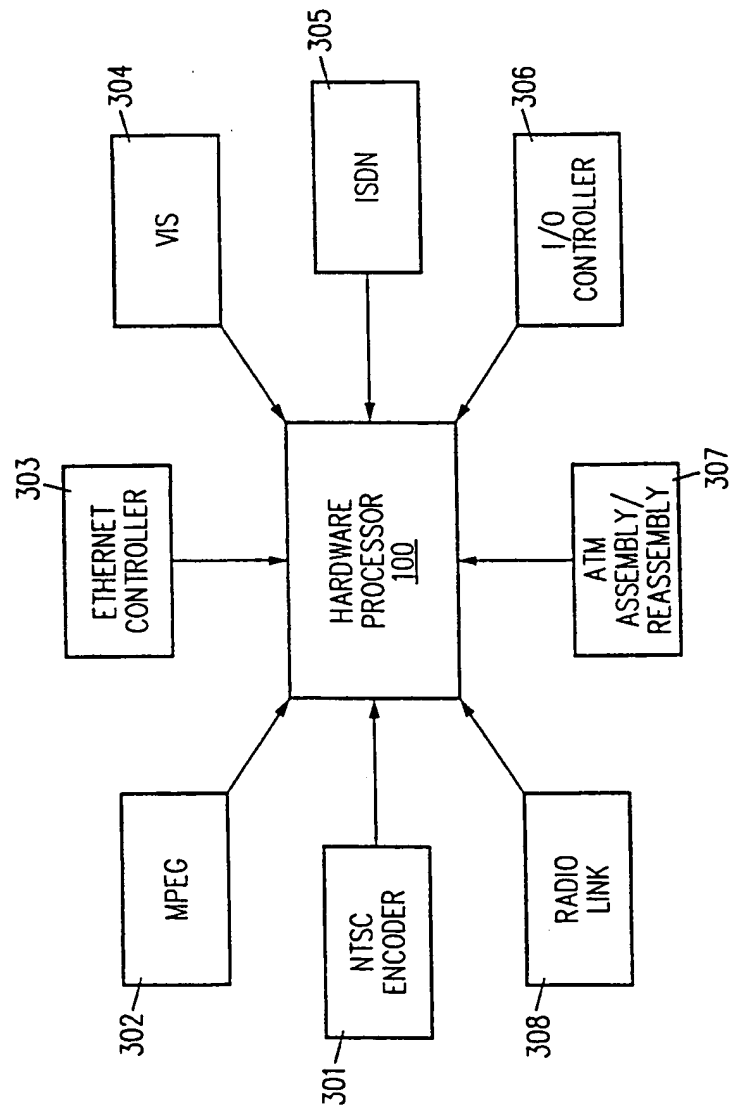


FIG. 3

5/17

FIG. 4A

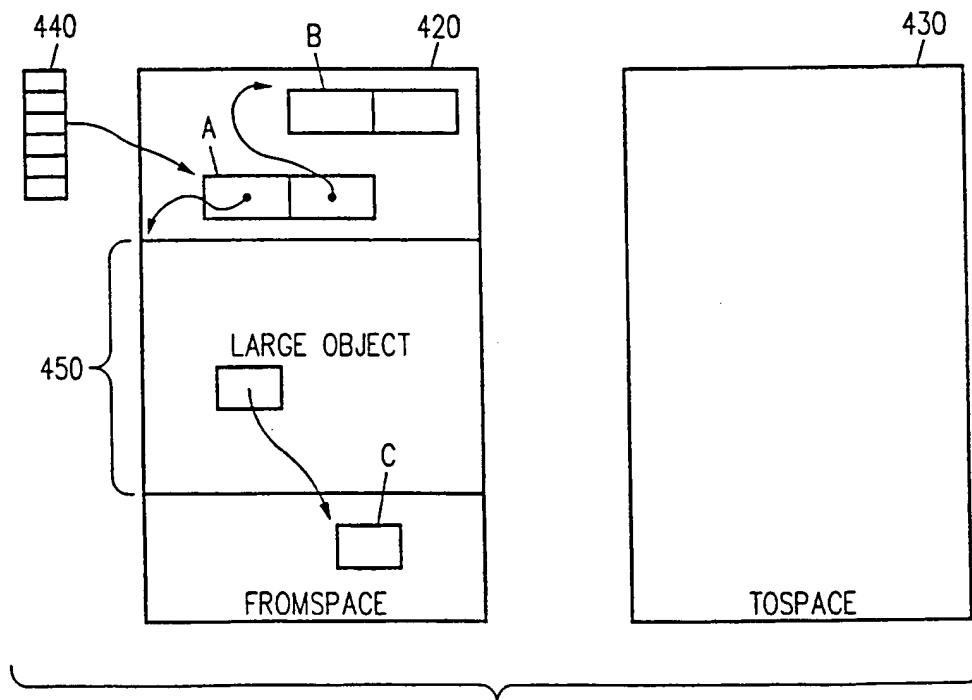
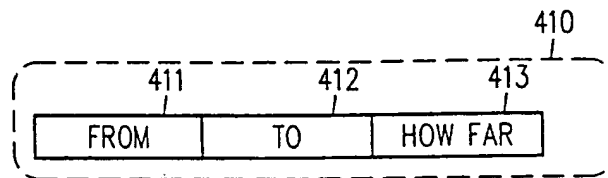


FIG. 4B

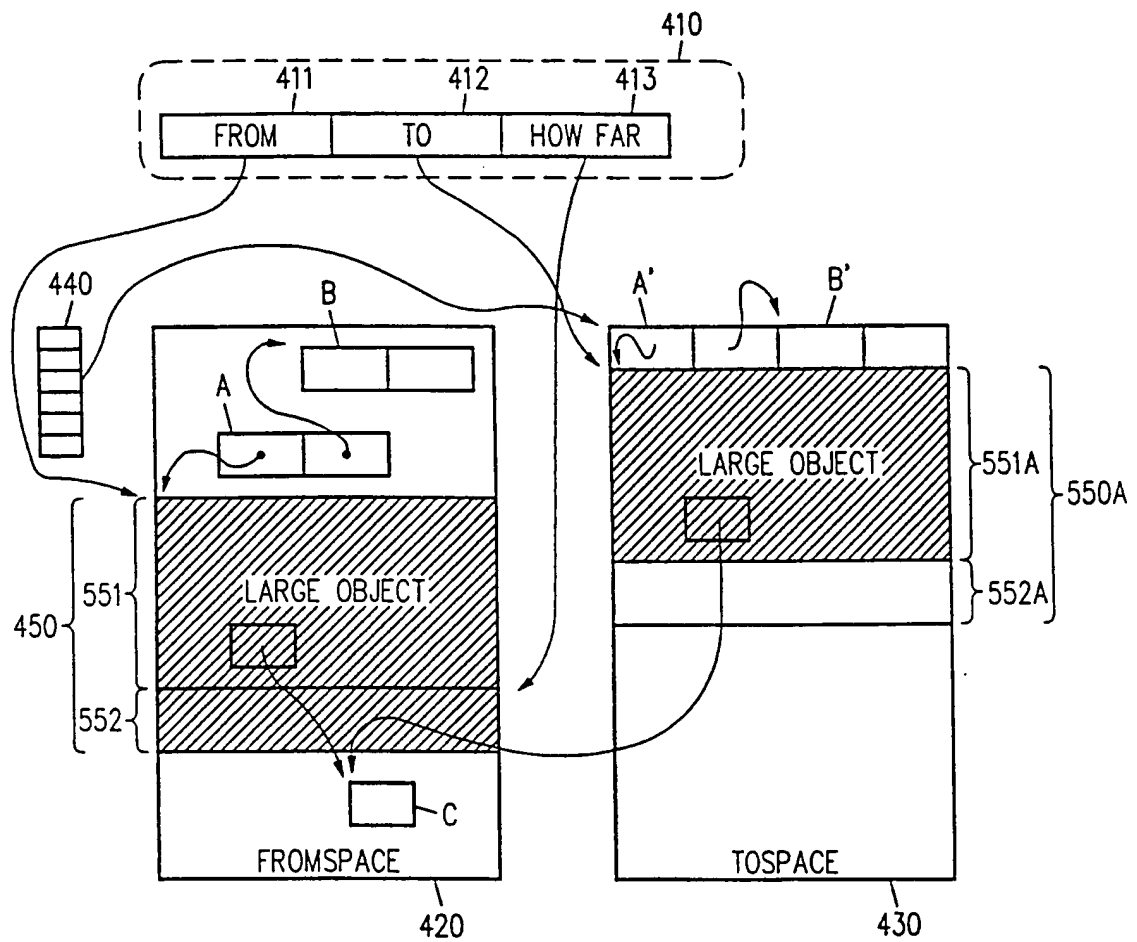


FIG. 5

7/17

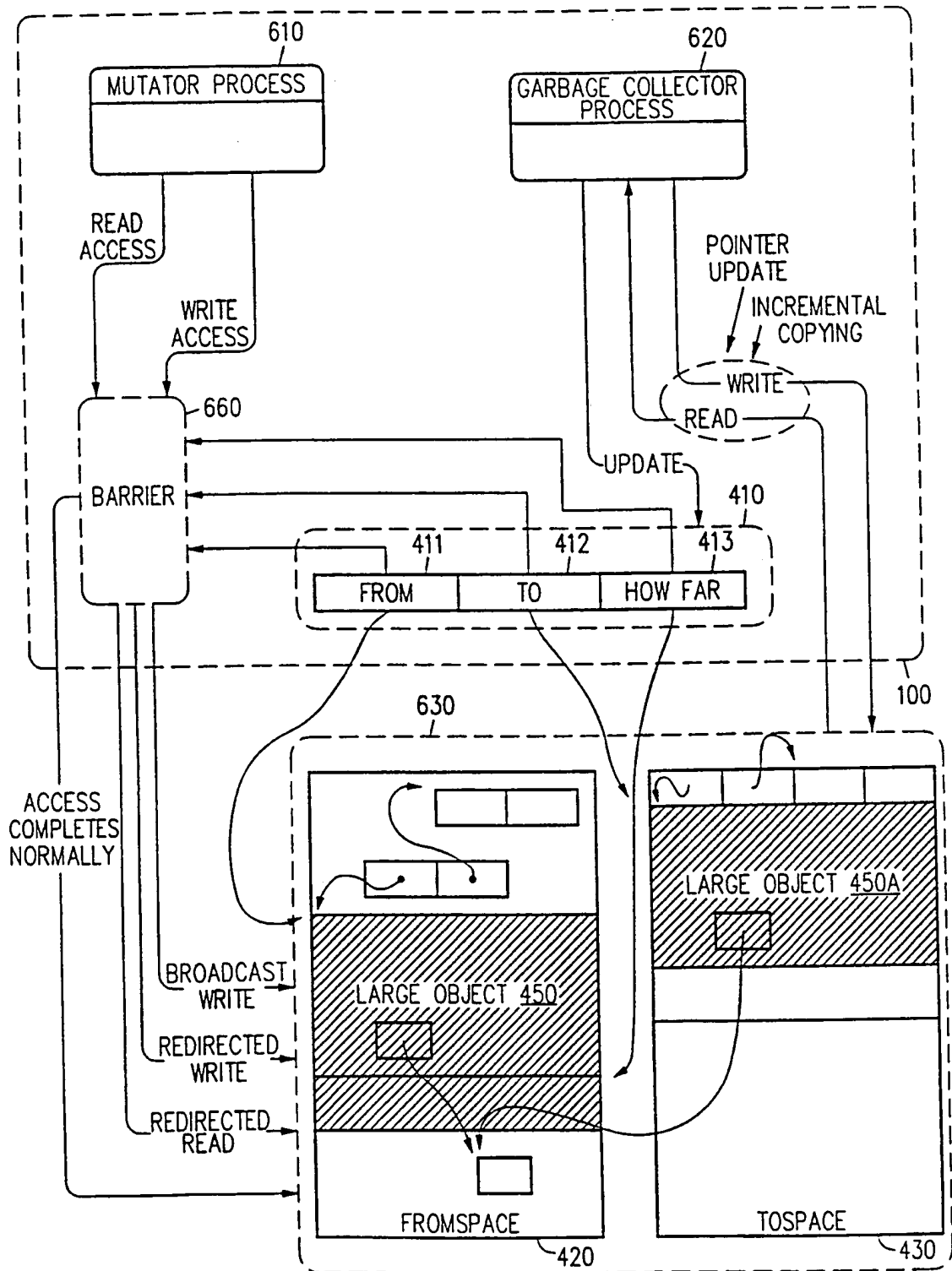


FIG. 6

8/17

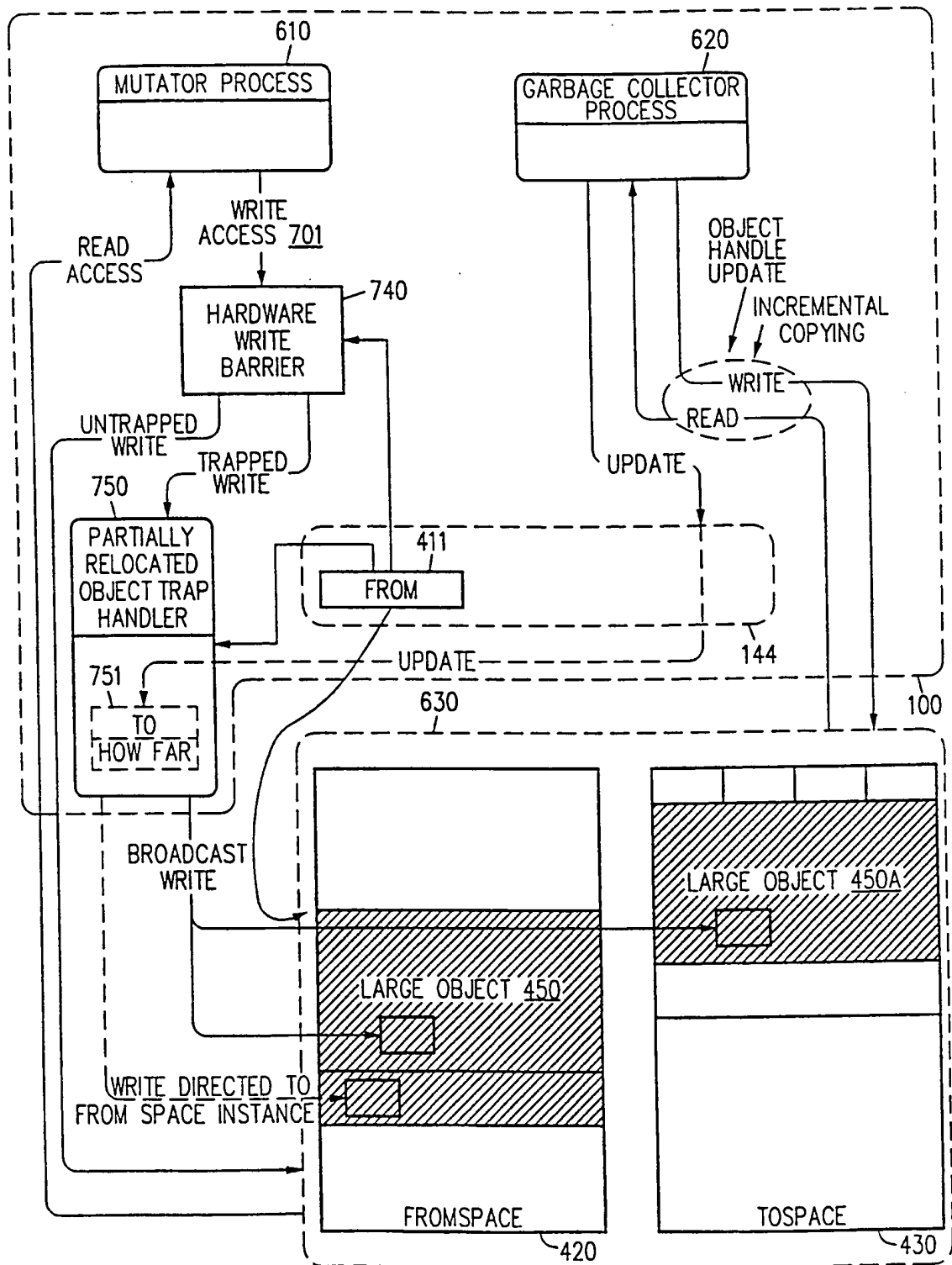


FIG. 7

9/17

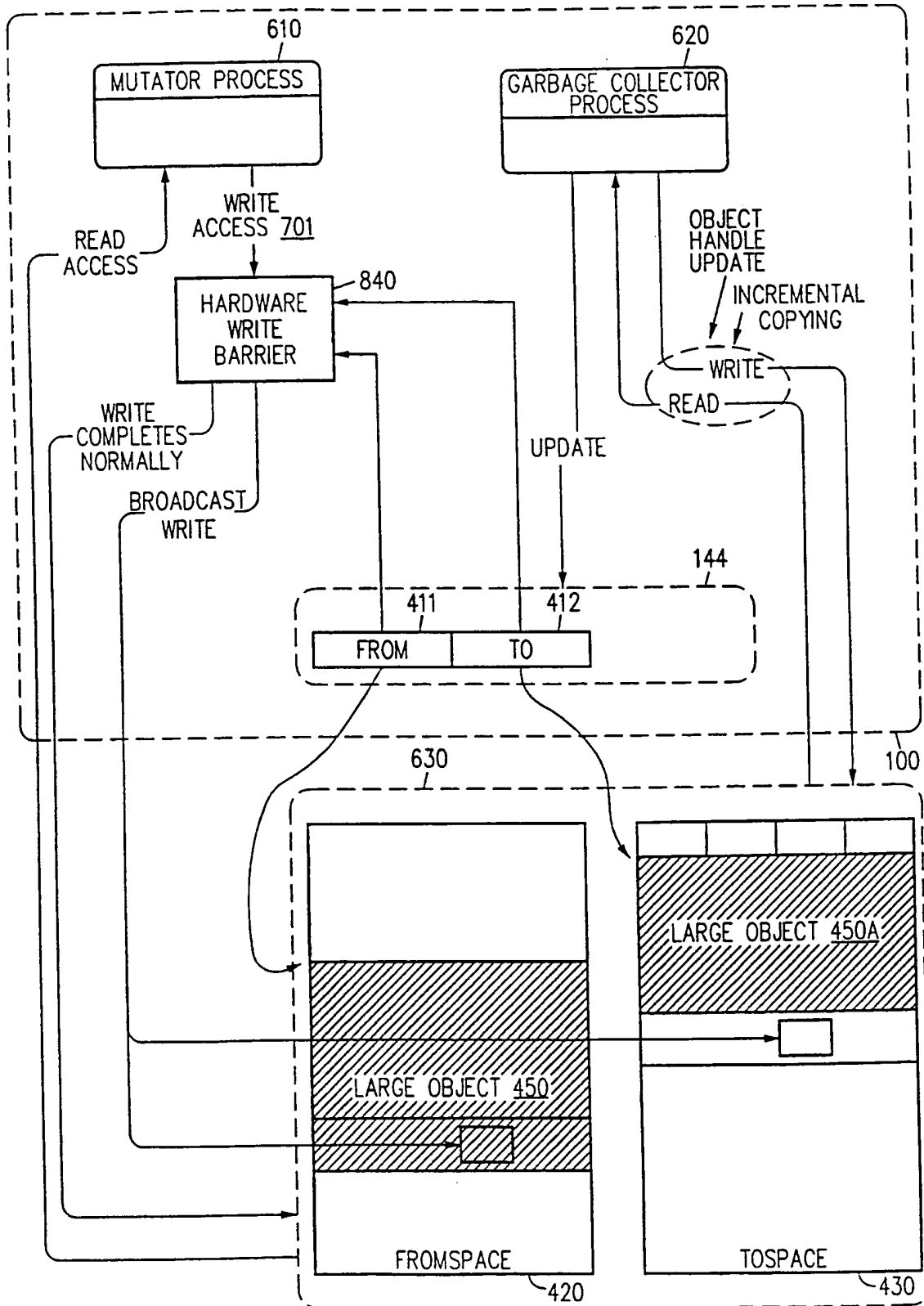


FIG. 8



10/17

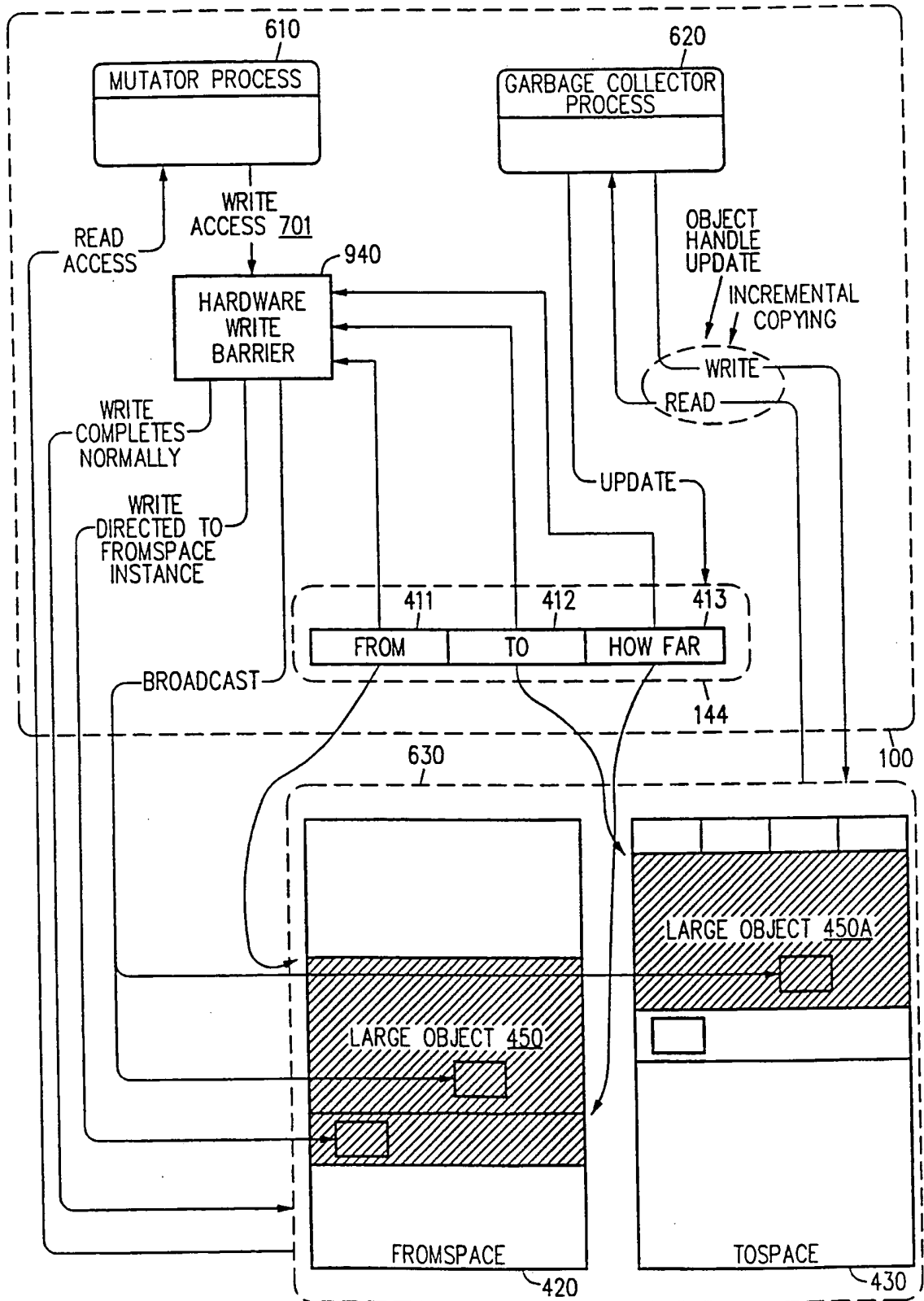


FIG. 9

11/17

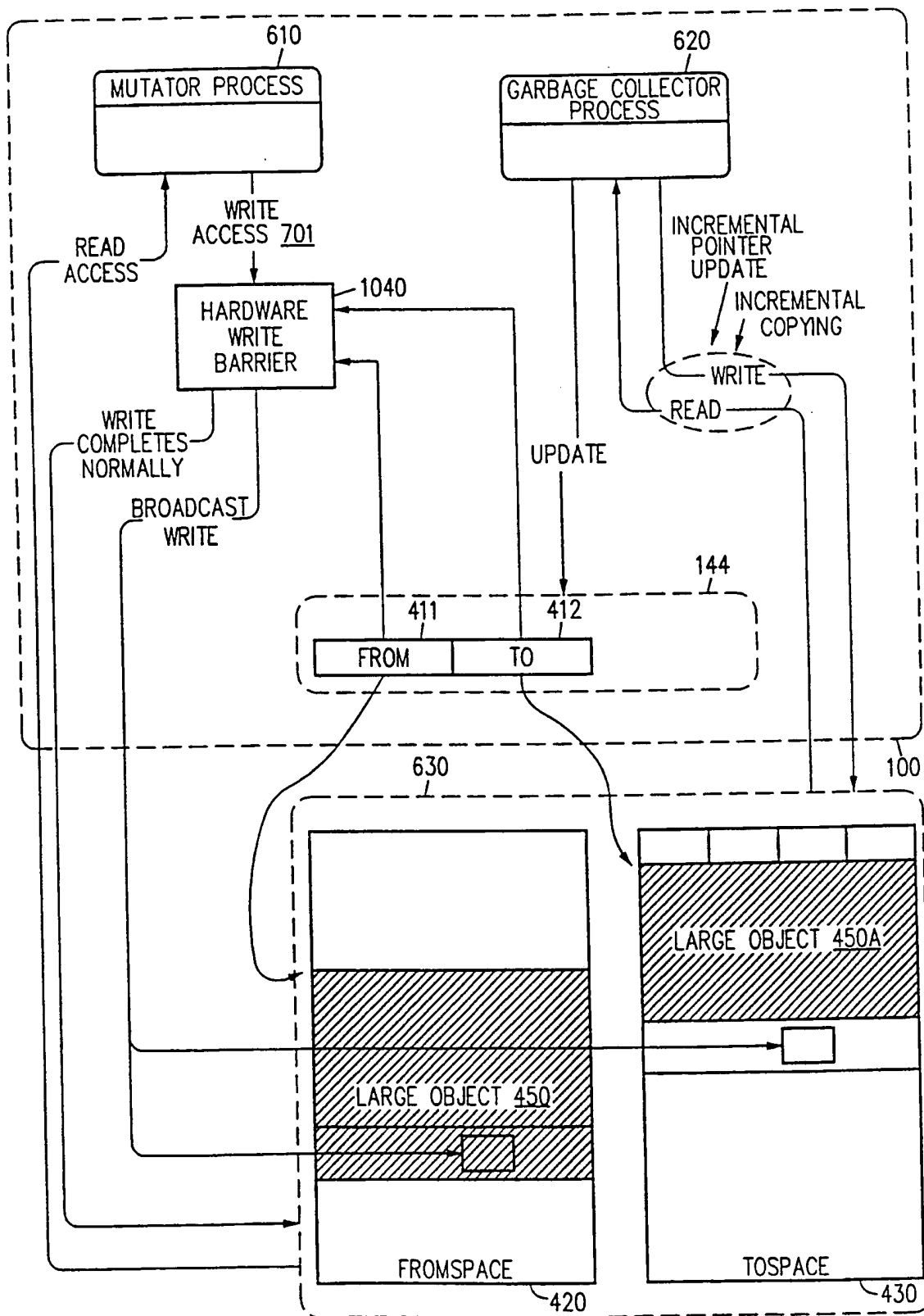


FIG. 10

12/17

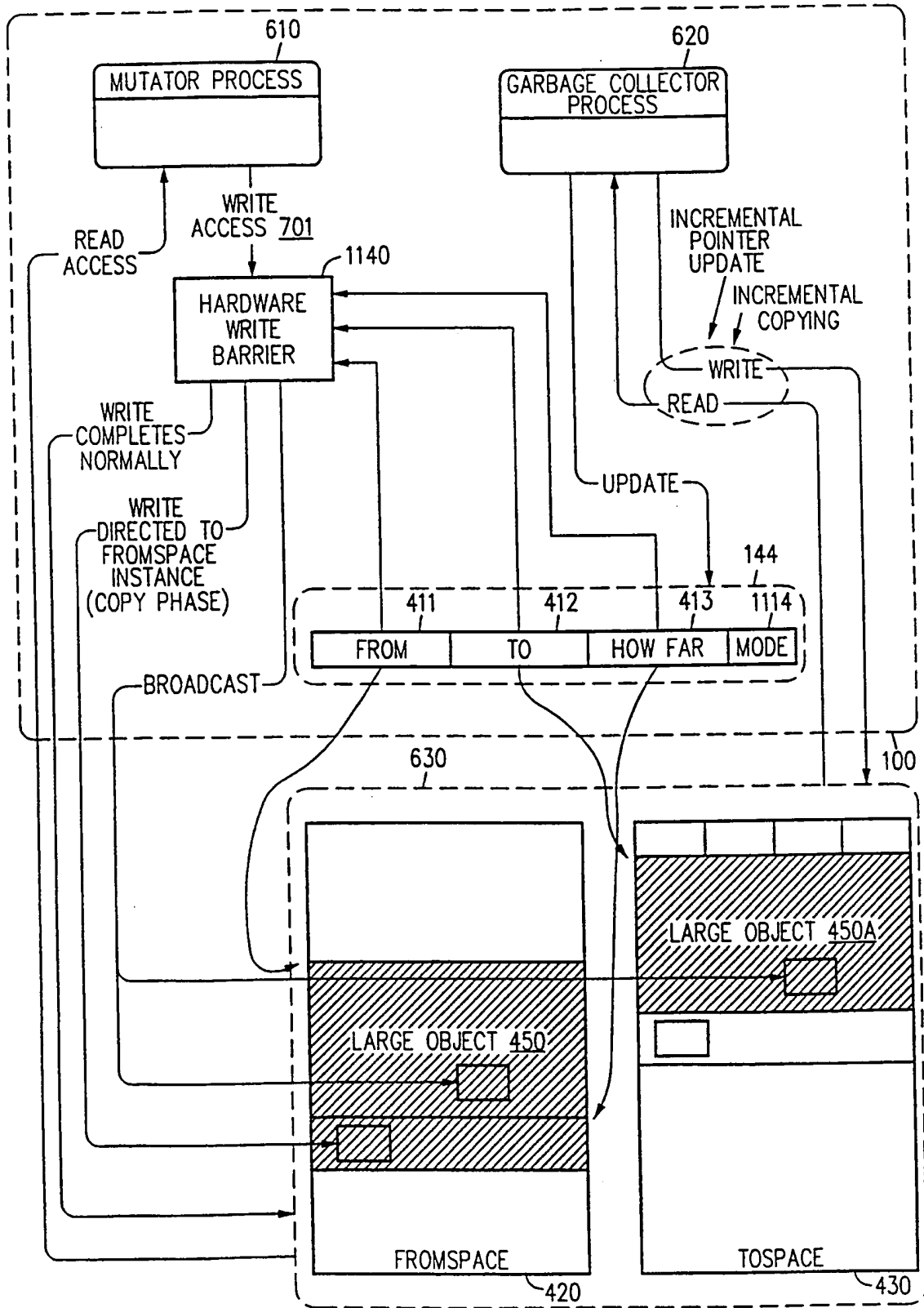


FIG. 11

13/17

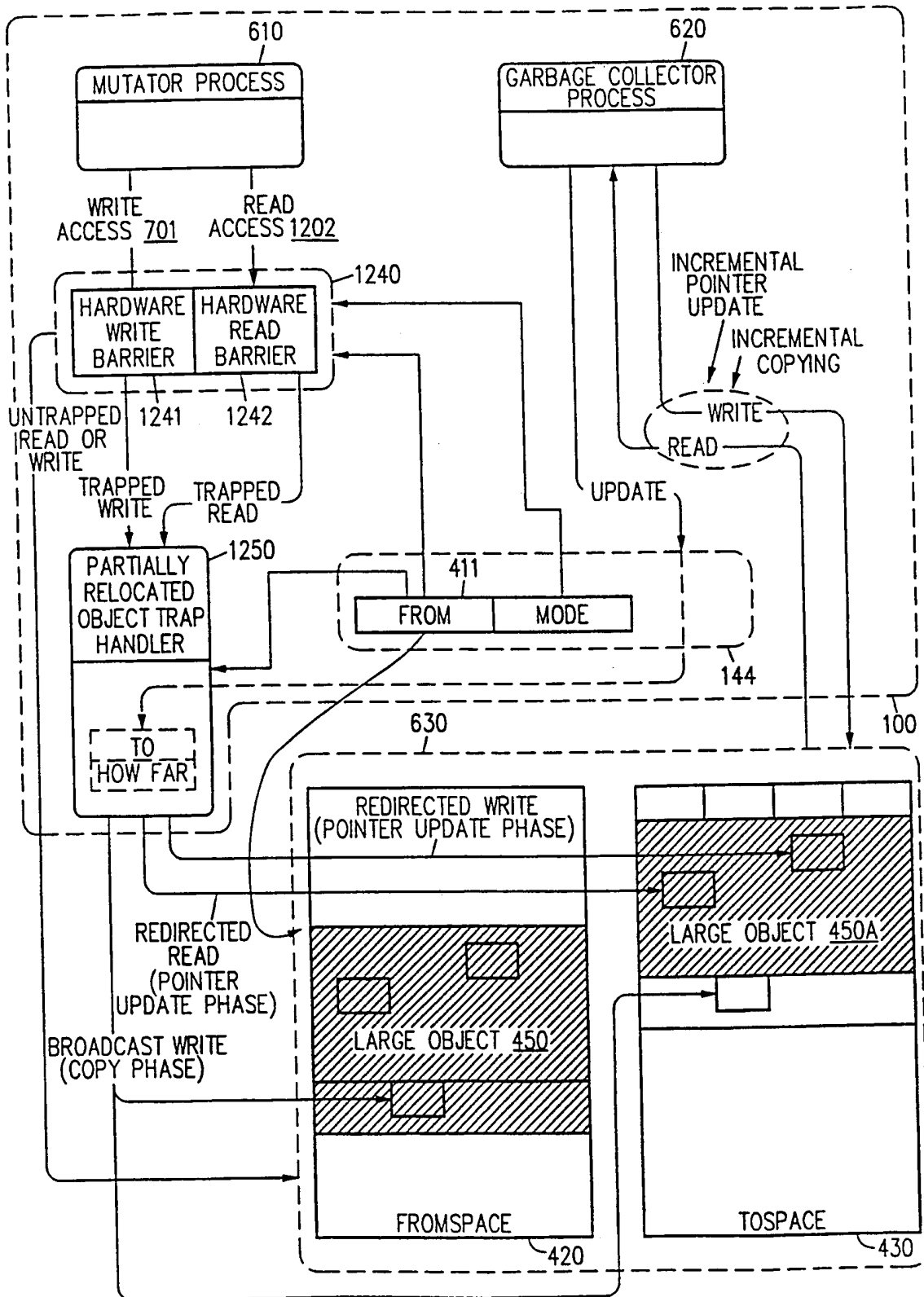


FIG. 12

14/17

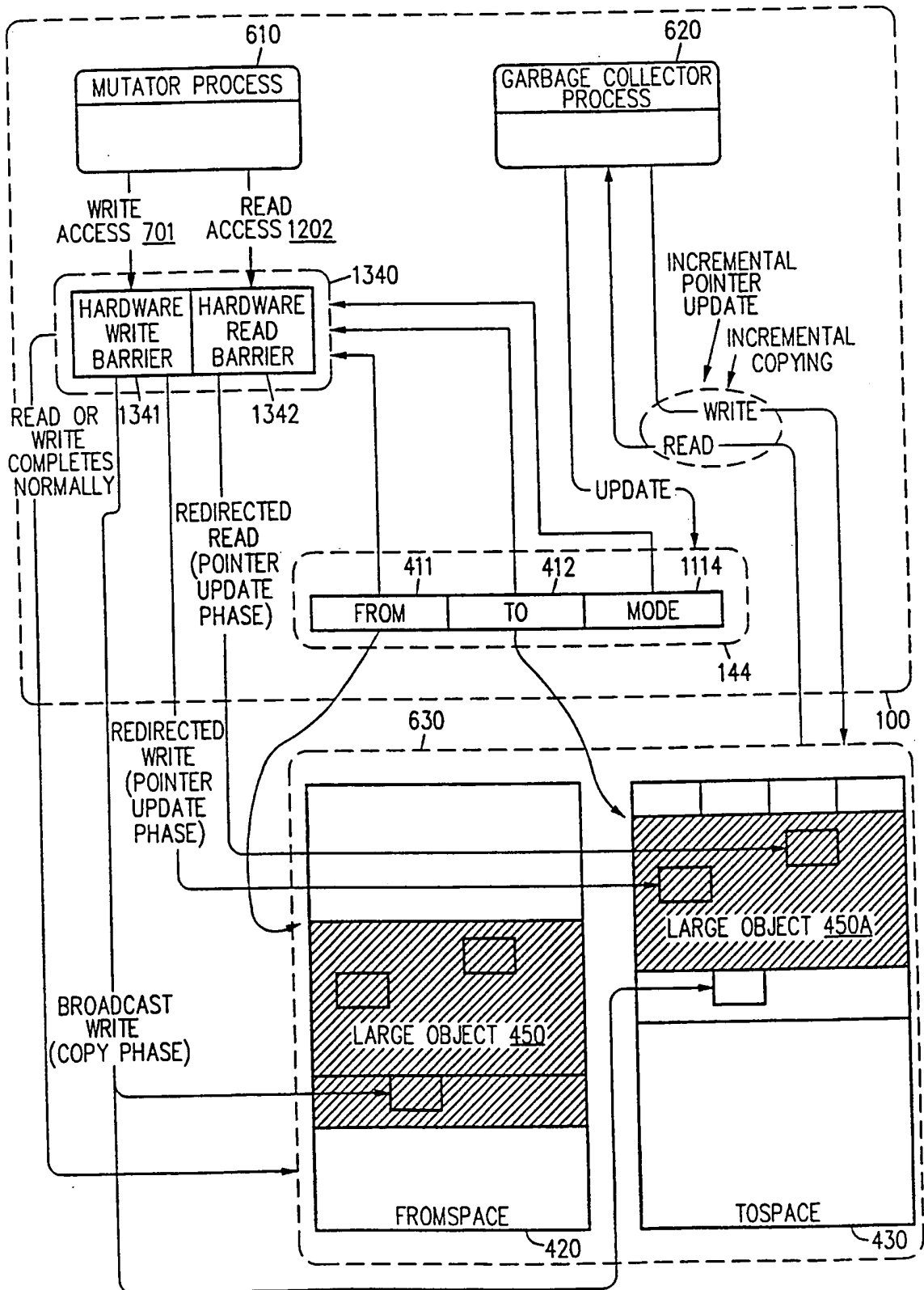


FIG. 13

15/17

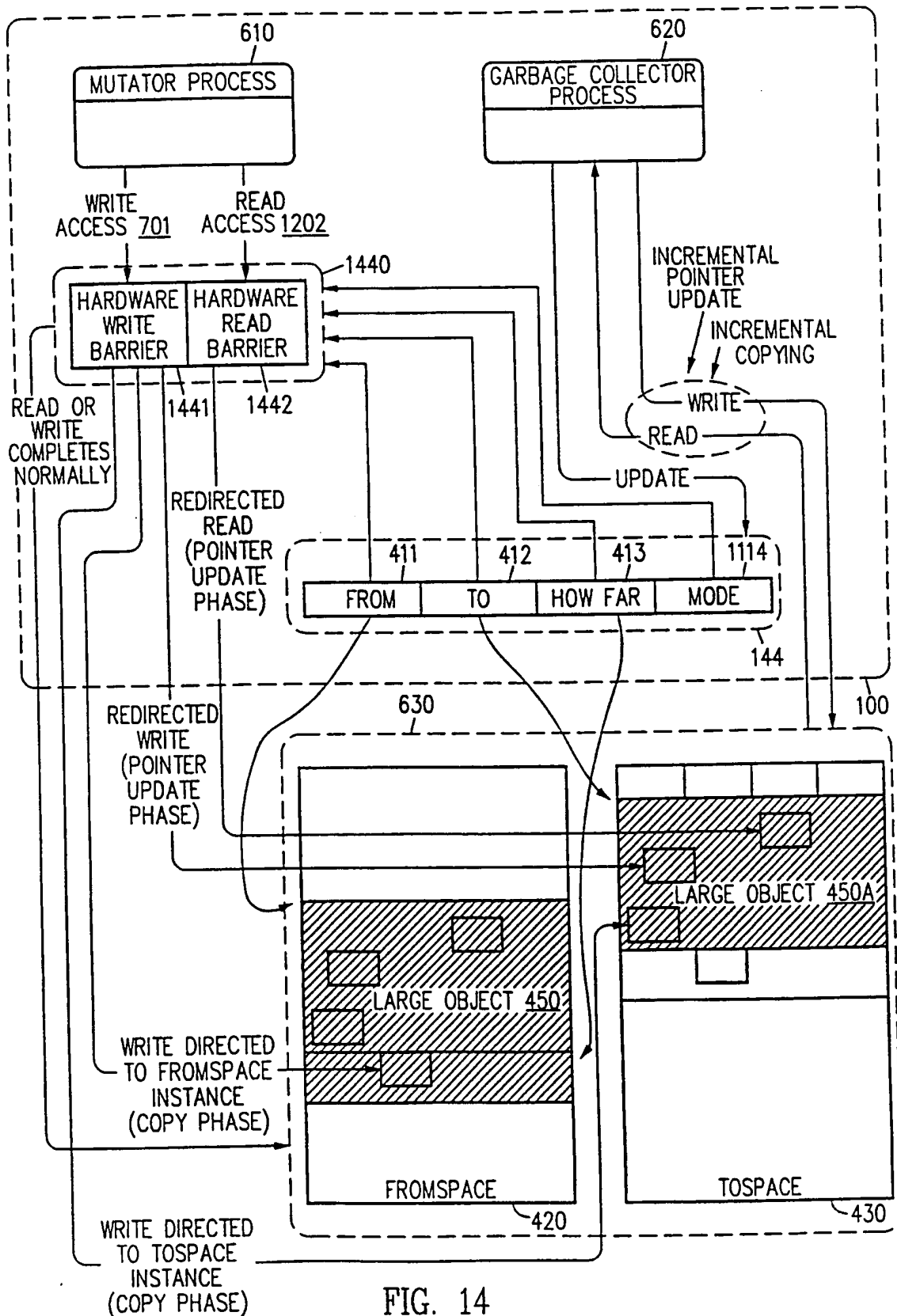


FIG. 14

16/17

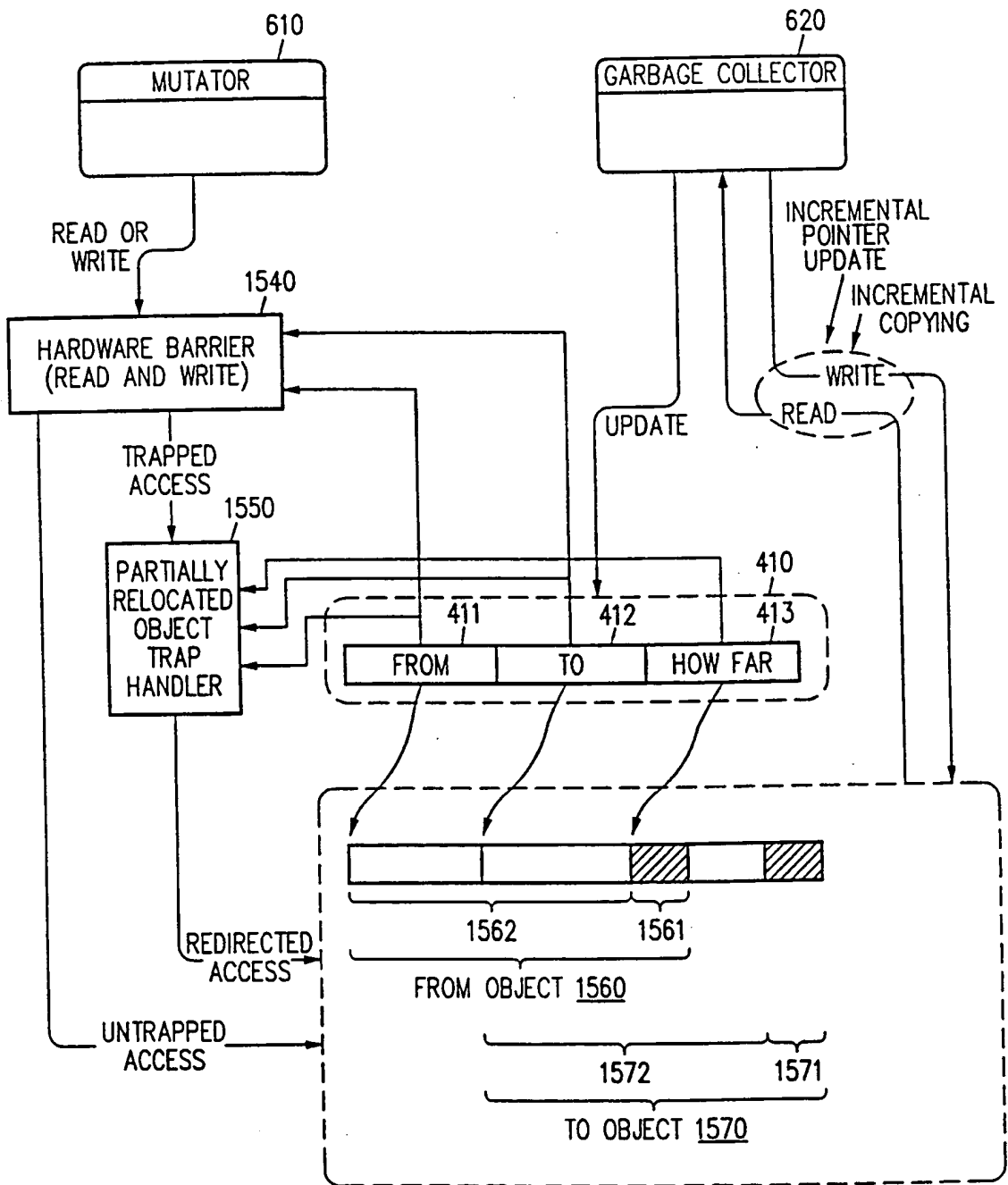


FIG. 15

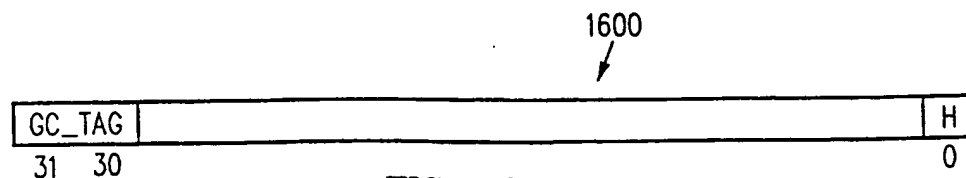


FIG. 16

17/17

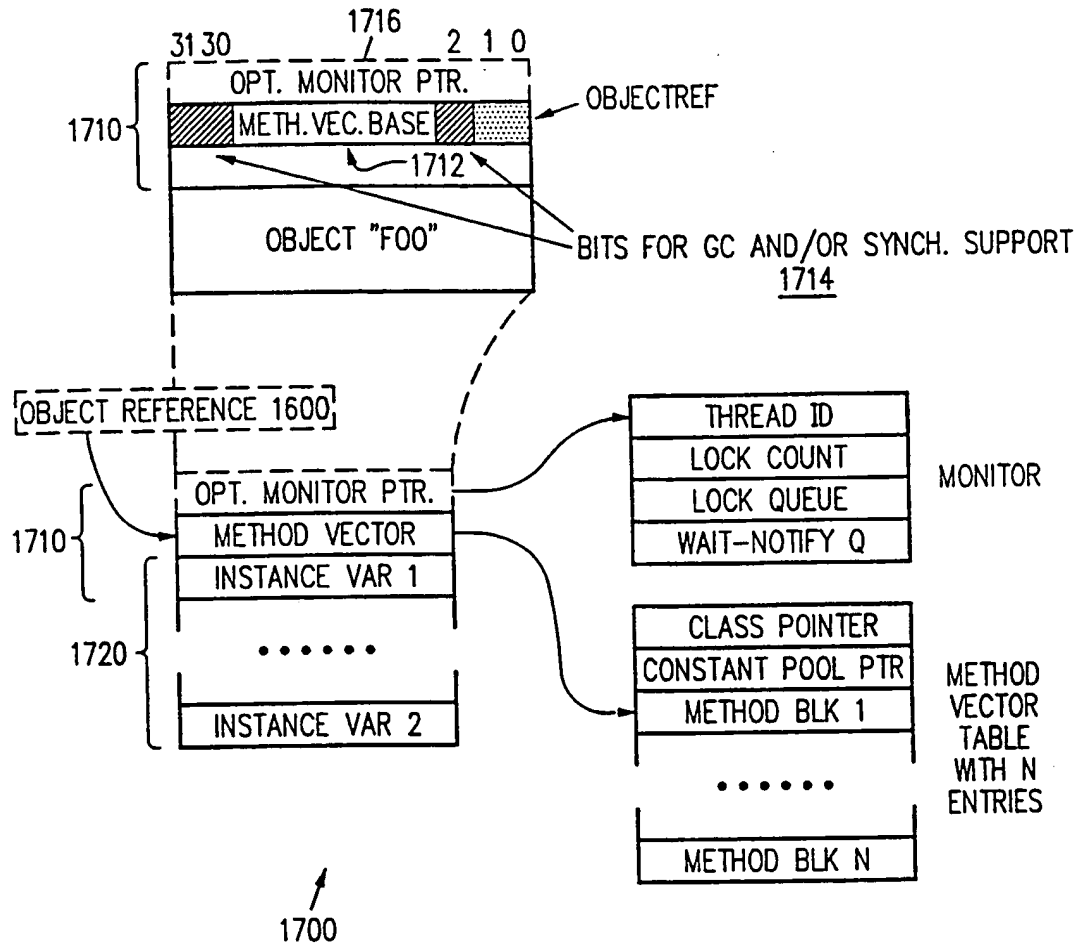


FIG. 17A

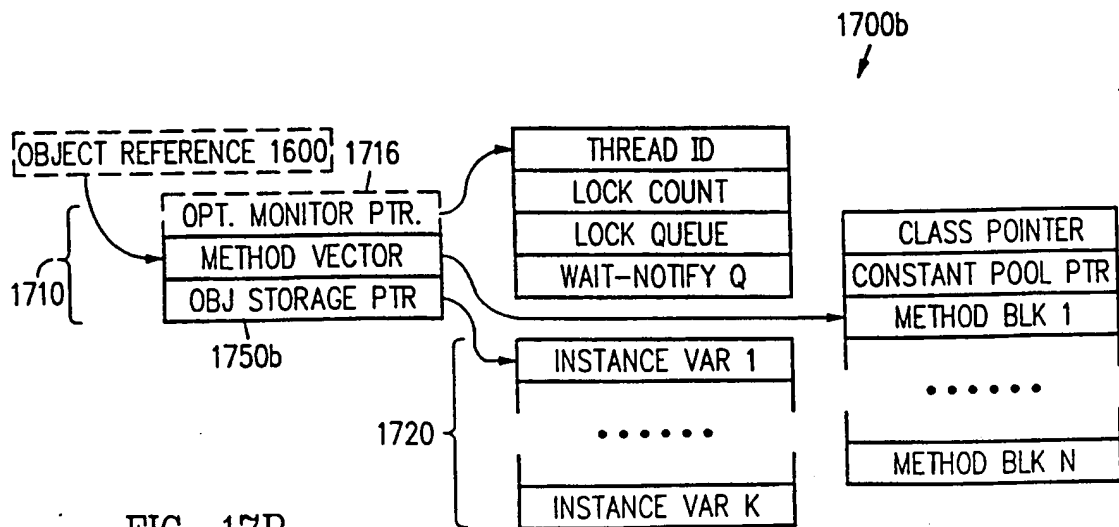


FIG. 17B



# INTERNATIONAL SEARCH REPORT

International Application No

PCT/US 98/13478

## A. CLASSIFICATION OF SUBJECT MATTER

IPC 6 G06F12/02

According to International Patent Classification(IPC) or to both national classification and IPC

## B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

## C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 5 560 003 A (NILSEN KELVIN D ET AL) 24 September 1996 cited in the application	1,25
A	see column 10, line 46 - column 16, line 14	2,3, 9-11,14, 15,28, 31,36,38
A	US 5 293 614 A (FERGUSON EDWARD E ET AL) 8 March 1994 see column 5, line 24 - column 8, line 38	1,25,28, 31,36,38



Further documents are listed in the continuation of box C.



Patent family members are listed in annex.

### \* Special categories of cited documents :

- \*A\* document defining the general state of the art which is not considered to be of particular relevance
- \*E\* earlier document but published on or after the international filing date
- \*L\* document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
- \*O\* document referring to an oral disclosure, use, exhibition or other means
- \*P\* document published prior to the international filing date but later than the priority date claimed

- \*T\* later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
- \*X\* document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
- \*Y\* document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.
- \*Δ\* document member of the same patent family

Date of the actual completion of the international search

25 September 1998

Date of mailing of the international search report

05/10/1998

Name and mailing address of the ISA

European Patent Office, P.B. 5818 Patentlaan 2  
NL - 2280 HV Rijswijk  
Tel. (+31-70) 340-2040. Tx. 31 651 epo nl,  
Fax: (+31-70) 340-3016

Authorized officer

Ledrut, P

# INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No

PCT/US 98/13478

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5560003 A	24-09-1996	US 5692185 A	25-11-1997
US 5293614 A	08-03-1994	NONE	